The 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2006

in co-operation with

EUROGRAPHICS

W S C G ' 2006

University of West Bohemia
Plzen
Czech Republic

January 31 – February 2, 2006

Full Papers Proceedings

Co-Chairs

Joaquim Jorge, Technical University of Lisboa,Lisboa, Portugal
Vaclav Skala, University of West Bohemia, Plzen, Czech Republic

Edited by
Joaquim Jorge, Vaclav Skala

WSCG'2006 Full Papers Proceedings

Editor-in-Chief: Vaclav Skala
University of West Bohemia, Univerzitni 8, Box 314
CZ 306 14 Plzen
Czech Republic
skala@kiv.zcu.cz
Managing Editor: Vaclav Skala

Author Service Department & Distribution:
Vaclav Skala - UNION Agency
Na Mazinách 9
322 00 Plzen
Czech Republic

Printed at the University of West Bohemia

Hardcopy: *ISBN 80-86943-03-8*

FOREWORD

This book contains the abstracts proceedings of WSCG'2006, the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2006. From its humble beginnings as the Winter School of Computer Graphics with about 20 participants from the former Czechoslovakia, in December 1992, it has become a focal point for Computer Graphics Community world-wide. Participants met in the period when the Czechoslovakia was to be split to two countries in January next year. During the past fourteen years, Computer Graphics has evolved into an exciting and diverse field that profoundly affects many aspects of everyday life.

As in previous editions, WSCG'2006 attracted the interest of the Computer Graphics Community, with a high number of paper submissions (296 in total), with contributions from many countries, from all continents which attest to the success and global dimension of the conference. From these, we selected 62 full papers, 28 short communication papers and 21 poster presentations. Each paper was reviewed by at least three experts selected among the 78 International Program Committee members. Complying with tradition, the best papers are published as a regular issue of the Journal of WSCG and all papers are available through the conference Digital Library at **http://wscg.zcu.cz/DL/wscg_DL.htm**. Also, published papers are listed on the major international scientific indexes such as Thomson ISI, which contributes to increasing the visibility of research work for authors. Furthermore, there were two keynote presentations from two distinguished speakers, Reinhard Klein from Bonn University and Gabriel Taubin from Brown University who presented comprehensive surveys and exciting talks addressing both the state of the art and future research directions.

Finally, we would like to express our thanks, first of all, to all the authors who submitted technical papers to WSCG'2006, and to the reviewers whose work and dedication made it possible to put together a very exciting program of high technical quality. We would also like to thank the invited speakers for their invaluable contribution and for sharing their vision in their talks.

The Program Co-Chairs

Joaquim Jorge, Technical University of Lisboa, Portugal
Vaclav Skala, University of West Bohemia, Plzen, Czech Republic

Plzen, January 21, 2006

WSCG 2006

International Programme Committee

Bartz,D. (Germany)
Bekaert,P. (Belgium)
Benes,B. (United States)
Bengtsson,E. (Sweden)
Bouatouch,K. (France)
Brunnet,G. (Germany)
Chen,M. (United Kingdom)
Chrysanthou,Y. (Cyprus)
Cohen-Or,D. (Israel)
Coquillart,S. (France)
Debelov,V. (Russia)
Deussen,O. (Germany)
du Buf,H. (Portugal)
Ertl,T. (Germany)
Ferguson,S. (United Kingdom)
Groeller,E. (Austria)
Hauser,H. (Austria)
Hege,H. (Germany)
Jansen,F. (Netherlands)
Jorge,J. (Portugal)
Kalra,P. (India)
Klein,R. (Germany)
Klosowski,J. (United States)
Kobbelt,L. (Germany)
Kruijff,E. (Germany)
Lars,K. (Sweden)
Magnor,M. (Germany)
Moccozet,L. (Switzerland)
Mudur,S. (Canada)
Mueller,K. (United States)
Muller,H. (Germany)
Myszkowski,K. (Germany)
OSullivan,C. (Ireland)
Pasko,A. (Japan)
Peroche,B. (France)
Post,F. (Netherlands)
Puppo,E. (Italy)
Purgathofer,W. (Austria)
Rauterberg,M. (Netherlands)
Rheingans,P. (United States)
Rokita,P. (Poland)
Rossignac,J. (United States)
Rudomin,I. (Mexico)
Sakas,G. (Germany)
Sbert,M. (Spain)
Schaller,N. (United States)
Schilling,A. (Germany)
Schneider,B. (United States)
Schumann,H. (Germany)
Shamir,A. (Israel)
Slusallek,P. (Germany)
Sochor,J. (Czech Republic)
Sumanta,P. (United States)
Szirmay-Kalos,L. (Hungary)
Taubin,G. (United States)
Teschner,M. (Germany)
Velho,L. (Brazil)
Veltkamp,R. (Netherlands)
Weiskopf,D. (Canada)
Westermann,R. (Germany)
Wu,S. (Brazil)
Wuethrich,C. (Germany)
Yamaguchi,F. (Japan)
Zara,J. (Czech Republic)
Zemcik,P. (Czech Republic)

WSCG 2006 Board of Reviewers

Adamo-Villani,N. (United States)
Adzhiev,V. (United Kingdom)
Ammon,L. (Switzerland)
Andreadis,I. (Greece)
Aran,M. (Turkey)
Araujo,B. (Portugal)
Aspragathos,N. (Greece)
Bartz,D. (Germany)
Batagelo,H. (Brazil)
Battiato,S. (Italy)
Bekaert,P. (Belgium)
Benes,B. (United States)
Bengtsson,E. (Sweden)
Beyer,J. (Austria)
Biber,P. (Germany)
Bieri,H. (Switzerland)
Bilbao,J. (Spain)
Biri,V. (France)
Bischoff,S. (Germany)
Borchani,M. (France)
Bottino,A. (Italy)
Bouatouch,K. (France)
Brodlie,K. (United Kingdom)
Brunnet,G. (Germany)
Buehler,K. (Austria)
Cohen-Or,D. (Israel)
Coleman,S. (United Kingdom)
Coquillart,S. (France)
Daniel,M. (France)
Danovaro,E. (Italy)
de Aguiar,E. (Germany)
De Decker,B. (Belgium)
Debelov,V. (Russia)
Del Rio,A. (Germany)
Deussen,O. (Germany)
Di Fiore,F. (Belgium)
Diaz,M. (Mexico)
du Buf,H. (Portugal)
Duce,D. (United Kingdom)
Erbacher,R. (United States)
Ertl,T. (Germany)
Feito,F. (Spain)
Felkel,P. (Czech Republic)
Ferguson,S. (United Kingdom)
Fernandes,A. (Portugal)
Flaquer,J. (Spain)
Fleck,S. (Germany)
Francken,Y. (Belgium)
Gagalowicz,A. (France)
Galo,M. (Brazil)
Geraud,T. (France)
Giachetti,A. (Italy)
Giegel,J. (United States)
Groeller,E. (Austria)
Gudukbay,U. (Turkey)
Guerreiro,T. (Portugal)
Habbecke,M. (Germany)
Haber,T. (Belgium)
Hanak,I. (Czech Republic)
Hast,A. (Sweden)
Hauser,H. (Austria)
Havran,V. (Germany)
Hege,H. (Germany)
Hernandez,B. (Mexico)
Herzog,R. (Germany)
Hirschbach,H. (Germany)
Hornung,A. (Germany)
Chen,M. (United Kingdom)
Chrysanthou,Y. (Cyprus)
Isgro,F. (Italy)
Jaillet,F. (France)
Janda,M. (Czech Republic)
Jansen,F. (Netherlands)
Jeschke,S. (Austria)
Jorge,J. (Portugal)
Jota,R. (Portugal)
Kalra,P. (India)
Kavan,L. (Czech Republic)
Keller,K. (United States)
Kipfer,P. (Germany)
Klein,K. (Germany)
Klosowski,J. (United States)
Kobbelt,L. (Germany)
Kolcun,A. (Czech Republic)
Krüger,J. (Germany)
Kruijff,E. (Germany)
Lanquetin,S. (France)
Lars,K. (Sweden)
Leon,A. (Spain)
Leopoldseder,S. (Austria)
Levy,B. (France)
Lintu,A. (Germany)
Linz,C. (Germany)
Lipus,B. (Slovenia)
Magalhaes,L. (Brazil)
Magillo,P. (Italy)
Magnor,M. (Germany)
Mantler,S. (Austria)
McMenemy,K. (United Kingdom)
Millan,E. (Mexico)
Moccozet,L. (Switzerland)
Moltedo,L. (Italy)
Montrucchio,B. (Italy)
Mudur,S. (Canada)
Mueller,K. (United States)
Muller,H. (Germany)
Multon,F. (France)
Myszkowski,K. (Germany)
Nielsen,F. (Japan)
Novotny,M. (Austria)
O'Sullivan,C. (Ireland)
Pasko,A. (Japan)
Patel,D. (Austria)
Patera,J. (Czech Republic)
Pedrini,H. (Brazil)
Perales,F. (Spain)
Peroche,B. (France)
Platis,N. (Greece)
Plemenos,D. (France)
Porcu,M. (Italy)
Post,F. (Netherlands)
Pratikakis,I. (Greece)
Prikryl,J. (Czech Republic)
Puig,A. (Spain)
Puppo,E. (Italy)
Purgathofer,W. (Austria)
Rauterberg,M. (Netherlands)
Reaz,M. (Malaysia)
Reina,G. (Germany)
Renaud,C. (France)
Revelles,J. (Spain)
Ribelles,J. (Spain)
Rodeiro,J. (Spain)
Rojas-Sola,J. (Spain)
Rokita,P. (Poland)

Rose,D. (Germany)
Rossignac,J. (United States)
Rudomin,I. (Mexico)
Sakas,G. (Germany)
Sanna,A. (Italy)
Sbert,M. (Spain)
Scateni,R. (Italy)
Segura,R. (Spain)
Shah,M. (United States)
Shamir,A. (Israel)
Schafhitzel,T. (Germany)
Schaller,N. (United States)
Scherzer,D. (Austria)
Schilling,A. (Germany)
Schneider,B. (United States)
Schneider,J. (Germany)
Scholz,V. (Germany)
Schumann,H. (Germany)
Sips,M. (Germany)
Sitte,R. (Australia)
Slusallek,P. (Germany)
Snoeyink,J. (United States)
Snoeyink,J. (United States)
Sochor,J. (Czech Republic)
Sojka,E. (Czech Republic)
Solis,A. (Mexico)
Sondershaus,R. (Germany)
Sporka,A. (Czech Republic)
Stephane,R. (France)
Stich,T. (Germany)
Strengert,M. (Germany)
Stroud,I. (Switzerland)
Stylianou,G. (Cyprus)
Sumanta,P. (United States)
Szekely,G. (Switzerland)
Szirmay-Kalos,L. (Hungary)
Tang,W. (United Kingdom)
Taubin,G. (United States)
Teschner,M. (Germany)
Theußl,T. (Austria)
Torres,J. (Spain)
Ulbricht,C. (Austria)
Van Laerhoven,T. (Belgium)
Vanecek,P. (Czech Republic)
Velho,L. (Brazil)
Veltkamp,R. (Netherlands)
Vergeest,J. (Netherlands)
Viola,I. (Austria)
VOLLRATH,J. (Germany)
Vuorimaa,P. (Finland)
Wan,T. (United Kingdom)
Weidlich,A. (Austria)
Weiskopf,D. (Canada)
Westermann,R. (Germany)
Wood,J. (United Kingdom)
Wu,S. (Brazil)
Wuethrich,C. (Germany)
Yilmaz,T. (Turkey)
Zach,C. (Austria)
Zachmann,G. (Germany)
Zalik,B. (Slovenia)
Zambal,S. (Austria)
Zara,J. (Czech Republic)
Zemcik,P. (Czech Republic)

WSCG 2006

Full Papers proceedings

ISBN 80-86943-03-8

Contents

(Additional files available on CD ROM version, only)

Dealing with Optical Material Properties in Computer Graphics and Vision

Reinhard Klein

Bonn University, Bonn
Germany

ABSTRACT

In the area of shape recognition tremendous advances were made during recent years. But in contrast to this success the recognition of materials is still a big challenge although for humans the identification of differnt materials is normally an easy task. In Computer Graphics we face a similar situation. While the modeling of shapes is already a highly-developed area the realistic modeling of the interaction of light with objects, i.e. the modeling of optical material properties, is still a great challenge. In addition, while there are straight forward techniques to measure differences between two shapes we need more sophisticated methods to judge the difference of reflection properties. From the rendering point of view these methods might even contain perceptual components.

One way to obtain realistic reflection properties are measurements of real world surfaces. For arbitrary (non-fluorescent, non-phosphorescent) materials, the reflection properties can be described by the 8D reflectance field of the surface, also called BSSRDF. Since densely sampling an 8D function is currently not practical various acquisition methods have been proposed which reduce the number of dimensions by restricting the acquisition to specific classes of materials. A subsequent data modeling step is performed to interpolate missing values and compress the measured data further.

In the first part of this talk we will give a brief overview over the different measurement techniques and algorithms used to capture reflection properties of different classes of objects targeted to the specific needs of computer graphics applications. Special emphasis will be given to surfaces with complex meso-structure.

In the second part we will discuss some techniques we used for the validation of the measurements and the resulting renderings. Strength and limitations of different aquisition and validation techniques will be discussed and future challenges will be identified.

Real-Time Massive 3D Data Capture and Geometry Processing

Gabriel Taubin

Brown University
USA

ABSTRACT

In Geometry Processing, a field which has developed during the last ten years, concepts from applied mathematics, computer science, and engineering are used to design efficient algorithms for the acquisition, manipulation, animation and transmission of complex 3D models. A number of methods have been proposed to smooth, denoise, edit, compress, transmit, re-parameterize, and animate very large polygon meshes, based on topological and combinatorial methods, signal processing techniques, constrained energy minimization, and the solution of partial differential equations. In particular, polygon models, which are used in most graphics applications, require considerable amounts of storage, even when they only approximate precise shapes with limited accuracy, and must be compressed by several orders of magnitude for fast network access. In this talk I will present some of our early contributions to the field, and some related ongoing research projects. I will also describe the state of our work towards the apparently unrelated goal of building Visual Sensor Networks with 1000s of cameras for real time capture and processing of 3D data.

Hierarchical topological structure for the design of a discrete modeling tool

Dexet M., Andres E.
SIC, Université de Poitiers
bât. SP2MI, Bvd Marie et Pierre Curie, BP 30179
86962 Futuroscope Chasseneuil Cedex, France
{dexet,andres}@sic.univ-poitiers.fr

ABSTRACT

In this paper, we present the design of a topological based geometrical modeler kernel. With this modeler, our goal is to create, import and handle geometrical objects represented both in a rasterized and a polygonal form. The aim is to provide a tool that mixes in a unified framework acquired discrete information (photos, MRI, ...) and synthetic continuous information (modeled objects). We propose the use of a multi-level hierarchical structure in which consecutive levels are linked. Each level of this structure corresponds to a particular representation of a same object. The lowest level is the raster representation and the highest level is the polygonal one. The way consecutive levels are computed, as well as the links between them are presented. Finally, we discuss the way the structure is updated using existing links in case of a modification applied on one level.

Keywords

Discrete geometry, geometrical and topological modeling, hierarchical structure, rasterization, polygonalisation.

1. INTRODUCTION

2D and 3D digital images are composed of discrete elements called *pixels* and *voxels*. Digital images can be generated, acquired (MRI, photos, ...) or can be the result of a rasterization (or *discretization*) process applied on a polygonal object (according to a discretization model). This data can be treated and manipulated using tools provided by *Discrete geometry*. One of the main problems we face when handling discrete information is that geometrical laws can be different in the discrete and in the continuous (Euclidean) world. For instance, some operations like rotations are simple in the Euclidean world while they present real difficulties in the discrete one. On the contrary, intersection of two objects is a simple operation in the discrete world and more difficult to perform in the Euclidean one. Having a discrete and a continuous representation of a same object can therefore have some interest. So far most existing modeling systems handle either vectorial based or raster information. Digital images can then be vectorized (using for instance 'Marching cubes' [LC87]) and polygonal objects can be rasterized, but no modeler proposes, to the best authors knowledge, to maintain both a polygonal and a raster representation of an object inside a same data structure. With this in mind, we present here a topology based modeling software that allows you to create, import or manipulate geometric objects in a continuous as well as in a discrete representation form. The particularity of this software is that discrete and continuous representations of a same object coexist inside a single hierarchical structure. It is thus possible, according to a given operation, to choose the most adapted representation. In a first approach [ABL01], objects where represented inside the modeling software with four different representations: continuous, discrete analytical, discrete contours and discrete. The discrete representations were all built based on the continuous one. Only the continuous representation could be modified and in this case, the other ones had to be entirely reevaluated. We now propose a new four level hierarchical structure. Each level corresponds to one of the previous representation forms. With this structure, each representation can be modified and all the others can be updated accordingly using reconstruction or discretization methods. The different consecutive levels in the hierarchical structure are

FULL Papers conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

linked. These links allow to manipulate and propagate modifications locally. Updating the whole structure is thus not systematically needed. Of course such an architecture comes with a prize. The complexity of the hierarchical structure is much more complex than classical topology based modeling or imaging softwares. Indeed, we introduce an architecture that aims at keeping a topological and a geometrical coherence between multiple representation forms. Finally, our structure is designed in any dimension and our modeler is based on a 3D topological structure (it is constructed as an extension of a 3D topology based Euclidean modeler). In the following, we briefly recall the definitions and properties of the topological and discrete analytical model that forms the basis of our structure. We also present the discrete analytical recognition and reconstruction processes.

Recalls on generalized maps

A topological model is used to explicitly specify adjacency and inclusion relationships between the different cells (vertices, edges, faces and volumes for dimensions 0, 1, 2 and 3) of a geometrical object. Information, called *embeddings*, such as geometrical ones (for instance vertex coordinates) can be added to the model. The kernel of our modeler is based on *generalized maps* [Lie94] which is a combinatorial model that allows the representation of cellular quasi-manifolds, orientable and non-orientable, with or without boundaries. A generalized map is a set of abstract elements, called *darts*, and applications on these darts. The definition of an n-dimensional generalized map, also called *n-G-maps*, is the following one:

DEFINITION 1 (n-G-MAP). *Let $n \geq 0$. An n-dimensional generalized map is $G = (D, \alpha_0, \alpha_1, \ldots, \alpha_n)$ where :*

- *D is a finite set of darts;*
- *$\forall i,\ 0 \leq i \leq n$, α_i is an involution[1] on D;*
- *$\forall i, j,\ 0 \leq i < i+2 \leq j \leq n$, $\alpha_i \circ \alpha_j$ is an involution.*

In an *n*-G-map, each topological cell of dimension *i* corresponds to a subset of darts called *orbit*. Such an orbit is denoted $<\alpha_0, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n>$ and corresponds to the set of darts that can be reached starting from a dart, using any combination of involutions except α_i. A cell of dimension i (or *i-cell*) is defined as follows:

DEFINITION 2 (*i*-CELL). *Let $G = (D, \alpha_0, \alpha_1, \ldots, \alpha_n)$ be an n-G-map, d a dart and $i \in N = \{1, \ldots, n\}$. The i-cell incident to d is the orbit $<>_{N-\{i\}}(d) = <\alpha_0, \ldots, \alpha_i - 1, \alpha_i + 1, \ldots, \alpha_n>(d)$.*

[1] An involution f on S is a one to one mapping from S onto S such that $f = f^{-1}$.

For example, in a 2-G-map, the orbits $<\alpha_1, \alpha_2>$, $<\alpha_0, \alpha_2>$ and $<\alpha_0, \alpha_1>$ correspond respectively to topological vertices, edges and faces. We can see in Figure 1 the decomposition of a topological object in cells. The object in Figure 1a can be decomposed into topological faces linked by an α_2 involution (see Figure 1b). In the same way, a face can be decomposed into topological edges linked by an α_1 involution (see Figure 1c). To finish, an edge can be decomposed into two darts which are linked by an α_0 involution (see Figure 1d).

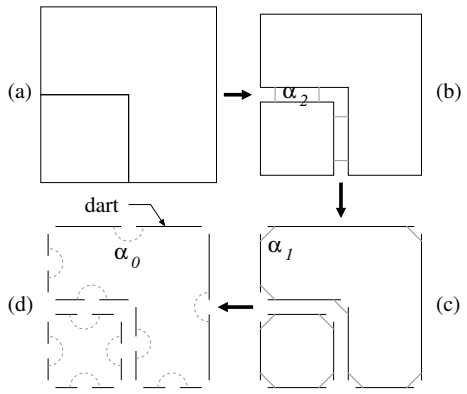


Figure 1: Topological decomposition of an object. (a) A 2D Euclidean object. (b) Its decomposition into topological faces. (c) Faces decomposition into topological edges. (d) Edges decomposition into darts.

In the following, in order to simplify figures, we will represent 2-G-maps as shown in Figure 2.

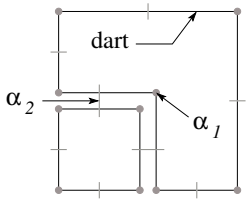


Figure 2: Representation of a 2-G-map.

The discrete standard analytical model

Our modeler allows to compute a discrete object from a given continuous object. In 2D (resp. 3D), the resulting discrete object must define 4-connected (resp. 6-connected) contours, in order to obtain a space subdivided in 4-connected (resp. 6-connected) regions. The discretization model we use is the *standard model* [And03]. We chose this model because it's the only model that allows the 4-connected analytical discretization of 2D objects (vertices and edges of polygons) and the 6-connected discretization of 3D objects (vertices, edges and faces of polyhedra). Moreover, this model is defined for any dimension.

In what follows, we focus on the 2D case. We first give the definitions of a discrete standard line and a discrete standard point. Then we explain how to compute the discrete analytical description of a segment. In [And03] the reader will find details about standard objects in 3D and higher dimensions.

DEFINITION 3 (2D STANDARD LINE). *A discrete standard line of parameters (a, b, c) is the set of integer points (x, y) verifying $-\omega \leq ax + by + c < \omega$, with $\omega = \frac{|a|+|b|}{2}$ and $a > 0$ or $a = 0$ and $b \geq 0$.*

DEFINITION 4 (2D STANDARD POINT). *Let P be a real point of coordinates (i, j). The standard discretization of P is the unique integer point (x, y) verifying $i - \frac{1}{2} \leq x < i + \frac{1}{2}$ and $j - \frac{1}{2} \leq y < j + \frac{1}{2}$.*

Figure 3 shows the standard discretization of a Euclidean segment (S in the figure).

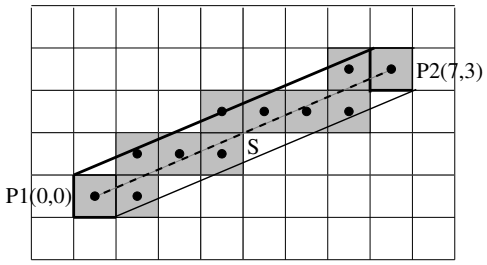


Figure 3: A standard segment and its analytical description. Pixels in gray correspond to the standard discretization of S (bold dotted line), i.e. pixels crossed by S. Bold (resp. thin) lines correspond to large (resp. strict) inequalities of the discrete analytical description.

The analytical description of S is deduced from the analytical descriptions of its end points $P1$ and $P2$, in addition to the analytical description of the line including S. These inequalities are:

- For the line: $-5 \leq 3x - 7y < 5$
- For P1: $-\frac{1}{2} \leq x < \frac{1}{2}$ and $-\frac{1}{2} \leq y < \frac{1}{2}$
- For P2: $7 - \frac{1}{2} \leq x < 7 + \frac{1}{2}$ and $3 - \frac{1}{2} \leq y < 3 + \frac{1}{2}$

The discrete analytical description of S is then composed of the following 6 inequalities (due to the removal of 4 inequalities):

$$\begin{cases} -5 \leq 3x - 7y < 5 \\ -\frac{1}{2} \leq x \\ -\frac{1}{2} \leq y \\ x < 7 + \frac{1}{2} \\ y < 3 + \frac{1}{2} \end{cases}$$

Note that two discrete spaces can be used to obtain discrete objects: the *classical discrete space* in which *discrete points* (i.e. integer coordinates points) are pixel centers (see Figure 4a), and the *dual discrete space* in which discrete points are pixel vertices (see Figure 4b). In our modeler, the discretization process is done in dual space, because we want to discretize the Euclidean region contours in order to obtain discrete region contours. The discretization of a polygon in the dual space is shown in figure 4b. Figure 4c shows the discretisation of two adjacent polygons.

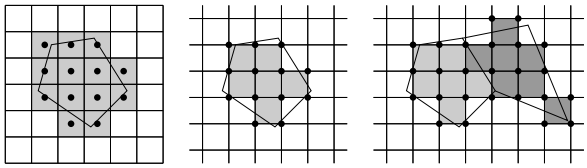


Figure 4: Standard discretization in the classical and dual discrete spaces. Black points are discrete ones. (a) Classical discrete space. (b) and (c) Dual discrete space.

Recognition and reconstruction processes

The discrete *recognition* process determines if a set of discrete points belongs or not to a given discrete object. The 2D recognition determines if a set of pixels belongs to a 2D discrete line. Figure 5a shows an example of discrete segment recognition. Adjacent pixels with the same color in the figure belong to the same discrete standard segment.

We call discrete *reconstruction* the operation that consists in obtaining a continuous object from a discrete one. This operation corresponds to the reverse of the discretization one, i.e. the discretization of the obtained continuous object corresponds to the original discrete object. Thus, the reconstruction of a 2D discrete segment is a Euclidean segment so that its discretization corresponds to the discrete segment. We see in Figure 5b an example of reconstructed segments.

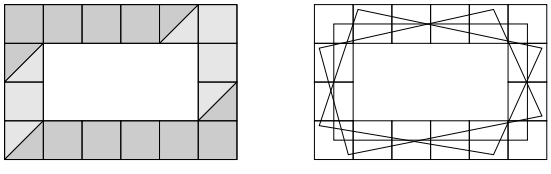


Figure 5: Discrete segment recognition and reconstruction. (a) Recognized discrete segments. Bicolor pixels belong to two discrete segments. (b) Three possible reconstructions for a same discrete curve.

Note that the reconstruction of an object is not unique (see Figure 5b). Indeed, an infinity of Euclidean objects have the same discretization. However, a reconstruction choice can be imposed, given a specific reconstruction method, choosing a starting point position (we choose the point with lowest abscissa and among

them the one with lowest ordinate) and a direction in which doing it (in our application, we choose the clockwise direction).
In the following, we describe our structure in dimension 2. In Sections 2 and 3, the different levels of the hierarchical structure are described, and the way each level is computed from a neighbor level is detailed. In particular, we focus on the computation of the links between two consecutive levels. Section 4 explains how those links are used to update the structure when a modification of a representation is performed. A short conclusion and perspectives are presented in Section 5.

2. MULTI LEVEL STRUCTURE DE SCRIPTION

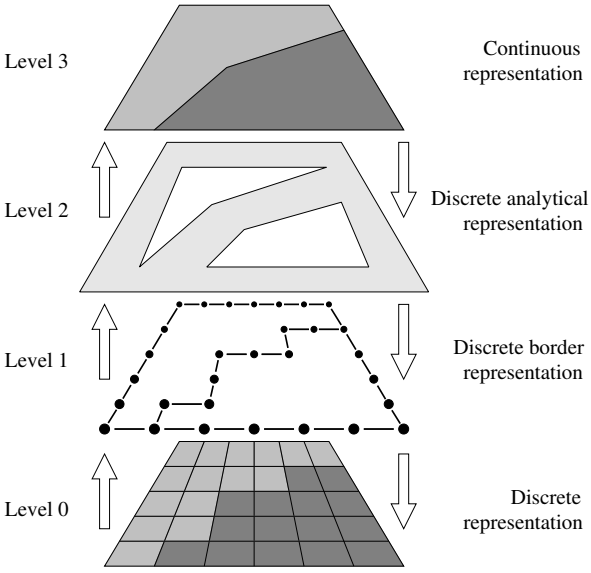


Figure 6: The hierarchical structure. Each level corresponds to a specific representation and is linked to its neighbors.

Our structure consists of four inter-dependent levels (see Figure 6): the discrete level, the discrete border level, the discrete analytical level and the continuous level. Each level corresponds to a particular representation of a same object. More precisely, a same primitive is represented either in a continuous form, and in three different discrete forms. Discrete and continuous levels are the end levels of the structure. The addition of two intermediate levels allows a progressive evolution from the discrete object to the Euclidean one, and vice versa. Each level of the structure is a generalized map. Information are associated with specific orbits of some levels in order to define the geometrical shape of the primitive.
In the following, we detail each level of this structure in dimension 2, i.e. its corresponding 2-G-map and embeddings. We illustrate our purpose by a simple example.

Level 0: the discrete level

This level corresponds to the classical discrete representation which is composed of pixels. The discrete space considered here is the dual one (see Section 1.2). The 2-G-map of this level corresponds to a space subdivision into pixels (see Figure 7a). Each pixel is represented by a square topological face associated with a color[2] embedding (see Figure 7b). Moreover, integer coordinates are attached to each topological vertex. The object in this level may be an imported image or the result of the discretization process applied on level 3.
However, the G-map of this level requires a lot of memory space, and particularly in 3D. For example, the memory space required for a 2D digital image of 512^2 pixels is at least 72 MB and for a 3D digital image of 512^3 voxels is 216 GB. In order to minimize the required memory space, the G-map of this level is coded in a more compact way as follows: embeddings are stored inside a matrix, and α_i involutions are not explicitly stored but recomputed from the neighborhood relations between the pixels. This is possible because pixels have a very simple topological form and adjacency relations are implicite. In the same way, connections between darts of level 0 and level 1 are simulated. This optimization brings the size of a 2D digital image of 512^2 pixels to less than 1 MB and the size of a 3D digital image of 512^3 voxels to 384 MB.

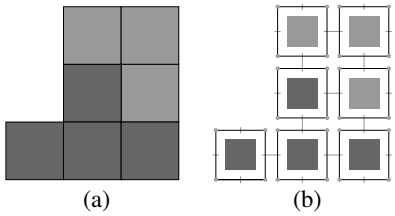
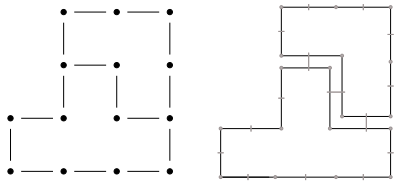


Figure 7: The discrete level. (a) Discrete view: here, a grayscale digital image. (b) Discrete 2-G-map: each topological face and each topological vertex are respectively associated with grayscale and integer coordinates embeddings.

Level 1: the discrete border level

This level corresponds to the contours obtained for each uniform colored 4-connected region. The representation of these contours is based on the *interpixel* model [KKM90, Kov89] (see Figure 8a). Each discrete point is represented by a point and two successive points are linked by a segment. This representation simplifies the coverage of level 0 regions boundaries. The 2-G-map of this level corresponds to a space subdivision into regions (see Figure 8b). There is no embedding at this level and geometrical information needed for the visualization of the level are located in level 0 and can be accessed from level 1 (see Section 4.1).

[2]This embedding can also be a label.

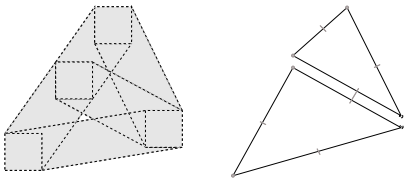


(a) (b)

Figure 8: The discrete border level. (a) Discrete border view. (b) Discrete border 2-G-map.

Level 2: the discrete analytical level

This level is an implicit representation of the discrete border primitive. It corresponds to the discrete analytical description of the level 1 region contours (see Figure 9a). More precisely, each contour is described as a discrete analytical polygon computed according to the discrete analytical standard model (see Section 1.2). The main interest of this representation is that it doesn't depend on the number of discrete points of the object. The 2-G-map of this level corresponds to a space subdivision into regions (see Figure 9b), in which each topological edge (resp. vertex) corresponds to a discrete analytical segment (resp. point). Each discrete segment (resp. vertex) is described by two (resp. four) opposite inequalities. These inequalities are associated to the topological edges and vertices of the level.



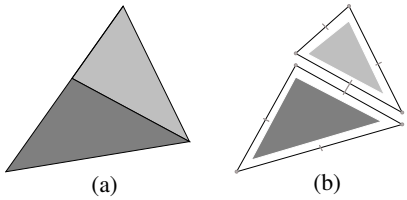
(a) (b)

Figure 9: The discrete analytical level. (a) Discrete analytical view. Gray areas are delimited by the analytical inequalities. (b) Discrete analytical 2-G-map. Inequalities are associated to each topological edge and each topological vertex.

Level 3: the continuous level

This level is an explicit representation. In this level, each region is described as a colored Euclidean polygon in the classical boundary representation form (see Figure 10a). The primitive of this level may be created using the tools available inside the modeler, or may be the result of the reconstruction process applied on level 0 (see Section 1.3). 2D Euclidean vertex coordinates and face colors are associated to the 2-G-map of this level (see Figure 10b).

For this level to be coherent with the others, we must ensure that the discretization (according to the standard model) of the Euclidean polygons is equal to the regions in level 1.



(a) (b)

Figure 10: The continuous level. (a) Continuous view. (b) Continuous 2-G-map with color and real coordinates embeddings.

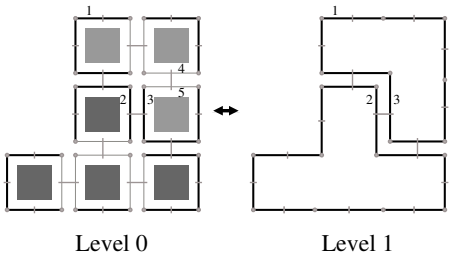
3. CONSTRUCTION AND LINKS BETWEEN THE LEVELS

Each level of the hierarchical structure corresponds to a step of the reconstruction or discretization process. In this section, we describe the way each level is built. To build the structure starting from level 0 or from level 3, topological operations are required. We won't describe them is this paper (see [DDAL05] for more details). Since our goal is to quickly reflect a modification of a level of the structure on the others, we set up bidirectional connections between the different levels, and more precisely between the darts of each level. The advantage of these connections is that all structure's embeddings become accessible for all levels. The information redundancy inside the structure is thus limited. In the next sections we explain how the different levels are built and connected.

Links between level 0 and level 1

- Level 1 is obtained from the discrete primitive of level 0 by merging pixels of same color.
- To build level 0 from level 1, we need to reconstruct the pixels, for example by using a flood-fill algorithm on a matrix corresponding to Level 1.

As shown in Figure 11, links are established between the darts of level 1 and the corresponding darts in level 0.



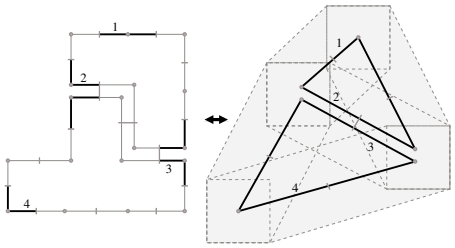
Level 0 Level 1

Figure 11: Links between level 0 and level 1. Black darts are connected ones. For example, darts numbered 1 (resp. 2 and 3) in the two representations are connected. Darts numbered 4 and 5 are not connected with darts of level 1. Here, there are exactly 36 links between the two levels.

Links between level 1 and level 2

- To obtain level 2 from level 1, we use an *analytical recognition* operation (see Section 1.3). The recognition algorithm we use is described in [BSDA03]. During the analytical recognition step, all vertices that belong to the same discrete segment are removed in order to keep only the two edge extremities. Removed darts are level 1 darts not connected with level 2 darts in Figure 12.
- To build level 1 from level 2, we need to discretize the contours of level 2 according to the standard model (see Section 1.2). Each edge of level 2 is incrementally discretized from one end point to the other, and a new vertex is inserted into the edge while the other end point is not reached.

For each edge of level 2, links are established between the darts of the edge and the extremity darts that belong to the corresponding discrete segment in level 1 (see Figure 12).



Level 1 Level 2

Figure 12: Links between level 1 and level 2. Black darts are connected ones. For example, darts numbered 1 (resp. 2, 3 and 4) in the two representations are together connected. Here, there are exactly 12 links between the two levels.

However, some problems can arise such as degenerated vertices (see Figure 13b). These vertices disappear in the discrete border level due to the loss of information inherent to the discretization process. We choose to delete these vertices (see Figure 13c), and move the links existing with these darts (numbered darts in Figure 13d) to keep the levels coherent.

Links between level 2 and level 3

- To build level 3 from level 2, we make a copy of level 2, and compute the Euclidean coordinates embeddings during the *analytical reconstruction* step (see Section 1.3). Reconstruction algorithms and details on particular cases that can occur during the analytical reconstruction are presented in [BSDA03] and [SBDA05].
- To build level 2 from level 3, we also make a copy of level 3, but we can eventually make some simplifications. For example, the gray square in Figure 14a contains vertices having the same standard discretization. All corresponding topological vertices, except the two extremities, can thus be removed (see Figure 14b).

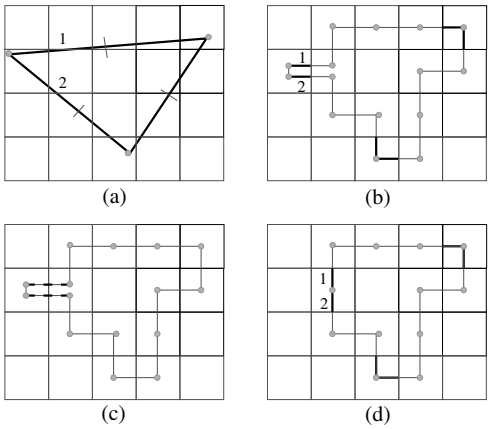


Figure 13: Example of degenerated vertex. (a) Level 2 primitive. (b) Primitive computed according to the standard discretization process. (c) Dashed darts are removed. (d) Level 1 primitive obtained. Note that black darts are connected ones.

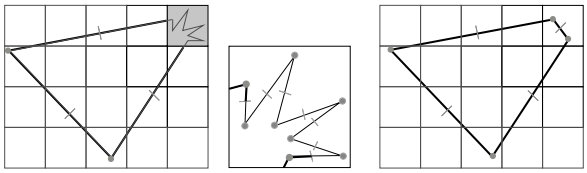


Figure 14: Simplification of the discrete analytical level. (a) Euclidean object. (b) The darts in gray are removed. (c) Discrete analytical level. Note that all black darts are connected ones.

Each dart of level 2 is connected with a dart of level 3 (see Figure 15). However, as in the example in Figure 14, some darts of level 3 may not be connected.

4. HIERARCHICAL STRUCTURE UP DATE

Information access

Stored information can be accessed using the links between consecutive levels. For example, integer coordinates stored at level 0 can be accessed from level 3 using successively the links between level 3 and level 2, the links between level 2 and level 1 and the links between level 1 and level 0. The main difficulty is to find linked darts between the current level and the following one. Indeed, all darts are not connected. For instance, some darts in level 1 are not connected with darts of level 2 (see Figure 12). Lets consider a face F of level 1. If we want to know the color associated to the face of level 3 corresponding to the reconstruction of F, we first must search for a connection between a dart of F and the

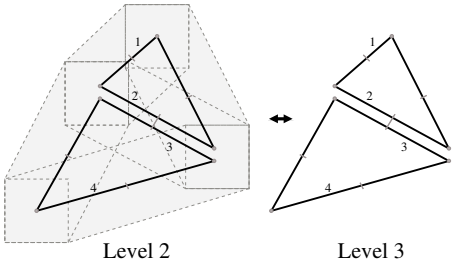


Figure 15: Links between level 2 and level 3. Black darts are connected ones. For example, darts numbered 1 (resp. 2, 3 and 4) in the two representations are together connected. On this example, there are exactly 12 links between the two levels.

corresponding face of level 2. The color embedding can then be reached using links between level 2 and level 3 since each dart of level 2 is connected with a dart of level 3.

Operations and updates on the structure

With our modeler, we can apply operations as well on the continous primitives as on the discrete ones. The most classical operation that can be used in level 0 is changing the color of one or several pixels. Many discrete operations can be based on it. Level 3 is based on a 3D Euclidean geometrical modeler called *Moka*[3] in which a lot of geometric operations have been already developed. Classical operations like translations, rotations, scales, ... or more elaborated operations like corefinement can be performed. In case of complex primitives (i.e. primitives composed of a great number of elements), updating all the structure consumes too much time. However, a level does not need to be totally recomputed when only a local modification is done. Moreover, it is often advisable to keep some details like those shown in figures 14a and 14b. In this case, a whole update of the structure will involve the loss of these informations. It is preferable to use the links between levels in order to reflect local modifications.

In our structure, when a level is modified by an operation, other levels may be updated. In this section, we show two different examples of operations on the structure: a discrete one and a continous one, and the way updates are computed.

4.2.1 Discrete operation: segmentation

The structure we consider here is based on a grayscale digital image (see Figure 16). We locally modify the segmentation of the digital image by filling the two middle regions in order to obtain only two regions. Thus, we can determine which darts must be removed in the other levels (darts connected – directly or not – with black ones in Figure 17). Indeed, in the original digital image, these darts where surrounded by two faces associated with different colors. After the edits, these darts are surrounded by two faces with the same color. Using the links between level 0 and level 1, we can easily find the darts of level 1 to be removed. And so on for level 2 and level 3. Here, the main interest of using links of the structure is that we can make local modification without recomputing the entire structure.

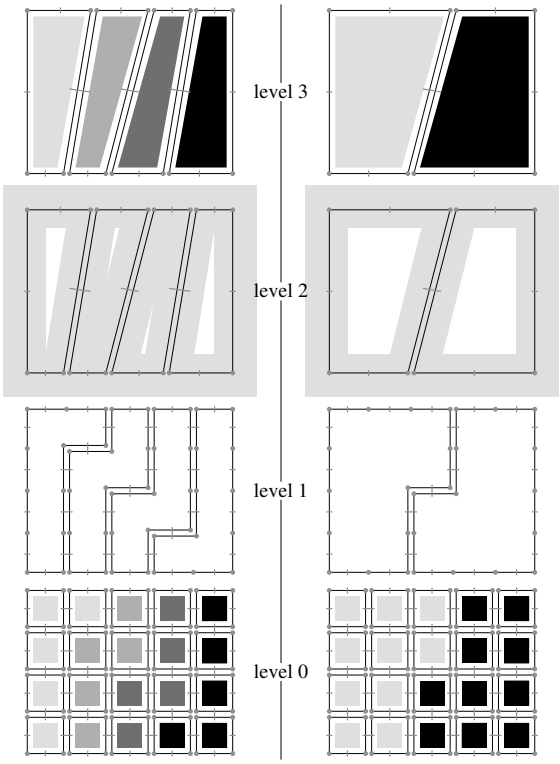


Figure 16: Our structure before and after color changes. On the left, the original digital image and the corresponding struture. On the right, the same structure after updates due to color changes.

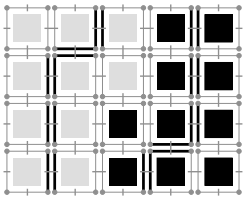


Figure 17: Darts concerned by the modification (in black).

4.2.2 Continuous operation: translation

The structure we consider here is based on a continous primitive (see Figure 18). We apply a non integer translation on level 3 (see Figure 19). We can see that the topology of the level 2 and 3 are not touched. Existing links between these two levels thus don't need to be changed. Nevertheless, we must modify the topology of the level 1 according to the discretization of level 2.

[3]Moka web site: http://www.sic.sp2mi.univ-poitiers.fr/moka

Some discrete segments can be kept if their discretization is still the same. Finally, level 0 has to be entirely recomputed (as well as links with level 1) because of the complexity of necessary local updates. We have thus seen that a global modification on level 3 may not require the structure to be entirely recomputed.

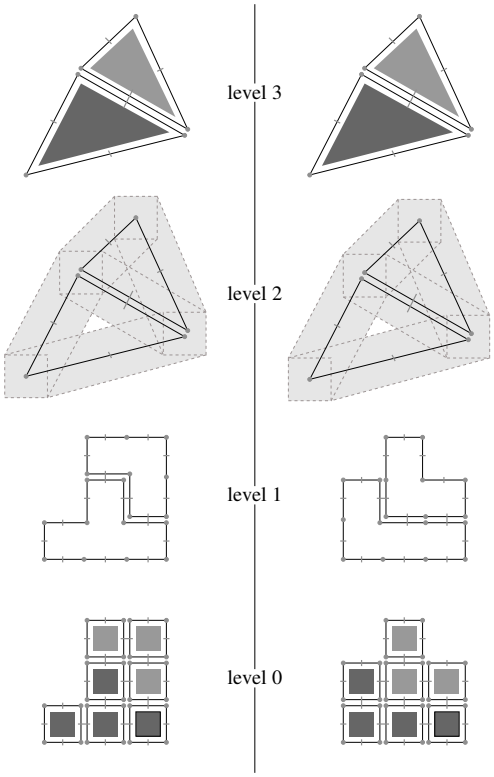


Figure 18: Our structure before and after level 3 primitive translation. On the left, the original Euclidean object and the corresponding structure. On the right, the same structure after updates due to the translation.

5. CONCLUSION

In this paper we have presented a modeler kernel based on a multi-level hierarchical structure. Each level corresponds to a particular representation of a same object: continuous, discrete analytical, discrete border of regions and discrete representations (see attached colored images). Each level is linked with the level above and below it. This ensures the coherence between all the representations. The way each level is computed from a neighbor one, as well as the way links are established between consecutive levels are presented. Two examples of operations on discrete and continuous levels are also detailed. As a short term goal, we plan to develop several operations and study more in details the propagation of local modifications inside the structure in order to optimize the updates even farther. Furthermore, a 3D extension is planned. Indeed, for the moment, our modeler deals only with 2D primitives because the 3D discrete analytical reconstruction operation, which aim is to provide a more compact reconstruction than the Marching cubes method, is still under developpement. A 3D version of our software should be available soon.

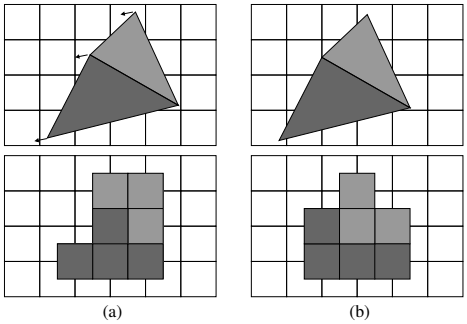


Figure 19: Discrete level before and after level 3 primitive translation. (a) Original continuous and discrete levels. (b) Continuous and discrete levels after translation.

6. REFERENCES

[ABL01] E. Andres, R. Breton, and P. Lienhardt. Spamod: design of a spatial modeling tool. *Digital and Image Geometry*, LNCS 2243:91–107, 2001.

[And03] E. Andres. Discrete linear objects in dimension n: the standard model. *Graphical Models*, 65:92–111, 2003.

[BSDA03] R. Breton, I. Sivignon, F. Dupond, and E. Andres. Towards an invertible euclidean reconstuction of a discrete object. In *Discrete Geometry for Computer Imagery*, volume LNCS 2886, pages 246–256, Naples, Italy, 2003.

[DDAL05] G. Damiand, M. Dexet, E. Andres, and P. Lienhardt. Removal and contraction operations to define combinatorial pyramids: application to the design of a spatial modeler. *Image and Vision Computing*, 23(2):259–269, 2005.

[KKM90] E. Khalimsky, R. Kopperman, and P.R. Meyer. Boundaries in digital planes. *Journal of Applied Mathematics and Stochastic Analysis*, 3:27–55, 1990.

[Kov89] V.A. Kovalevsky. Finite topology as applied to image analysis. *CVGIP*, 46:141–161, 1989.

[LC87] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *SIGGRAPH*, volume 21 of *Computer Graphics J.*, pages 163–169, Anaheim, USA, 1987.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3), 1994.

[SBDA05] I. Sivignon, R. Breton, F. Dupond, and E. Andres. Discrete analytical curve reconstruction without patches. *Image and Vision Computing*, 23(2):191–202, 2005.

Dual Subdivision

A New Class of Subdivision Schemes Using Projective Duality

Hiroshi Kawaharada and Kokichi Sugihara
Department of Mathematical Informatics Graduate School of Information Science and Technology, University of Tokyo, Japan
kawarada@simplex.t.u-tokyo.ac.jp,sugihara@mist.i.u-tokyo.ac.jp

ABSTRACT

This paper proposes a new class of subdivision schemes. Previous subdivision processes are described by the movement and generation of vertices, and the faces are specified indirectly as polygons defined by those vertices. In the proposed scheme, on the other hand, the subdivision process is described by the generation of faces, and the vertices are specified indirectly as the intersections of these faces. In this sense, this paper gives a framework for a wide class of new subdivision methods. In short, the new subdivision is a dual framework of an ordinary subdivision based on the principle of duality in projective geometry. So, the new subdivision scheme inherits various properties of the ordinary subdivision schemes. In this paper, we define the dual subdivision and derive its basic properties.

Keywords: subdivision surface, dual schemes, smoothness analysis

1 INTRODUCTION

Subdivision [11, 13, 3, 22, 20] is a well-known method for geometric design and for computer graphics, because the subdivision makes smooth surfaces with arbitrary topology. A subdivision scheme is defined by subdivision matrices and a rule of connectivity change. So, many researchers study the condition of continuity of subdivision surfaces depending on subdivision matrices [22, 24, 17, 16, 1, 7, 23, 18, 4]. Moreover, multiresolution analysis [14, 21, 5] derived by subdivision theory is extremely useful on mesh editing.

Subdivisions on quadrilateral or triangular meshes were studied extensively. For example, the most popular subdivisions are the Catmull-Clark subdivision and the Loop subdivision [13]. These subdivisions are designed for irregular quadrilateral or triangular meshes.

Most subdivision methods are for triangular or quadrilateral meshes; there are only a few methods for other types of meshes. Some subdivisions on hexangular meshes were developed [2, 6]. However, faces generated by these subdivisions are not "flat".

In this paper, we derive new subdivision schemes. This is the dual framework of an ordinary subdivision based on the principle of duality in projective geometry. The proposed dual subdivision can generate meshes, composed of non-triangular "flat" faces.

WSCG 2006 conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 - February 3, 2006
Plzen, Czech Republic.

Approximating surfaces using non-triangular flat faces is a basic problem for computational geometry. If the surface is convex, the approximation is easy. However, if the surface is not convex, the approximation can not be completed yet [19]. This dual subdivision schemes overcome this problem.

The dual subdivision is a wide class of new subdivisions. The dual subdivision generates faces by subdivision matrices. The subdivision matrices are shared between an ordinary subdivision and the corresponding dual subdivision. So, the dual subdivision has properties similar to the ordinary subdivision. In this paper, we explain such properties.

Levin and Wartenbarg [12] already proposed some dual schemes. However, their schemes are convexity-preserving interpolations. In first place, dual subdivision can naturally represent surfaces with arbitrary topology. Our framework enables it by "inflection plane".

From the mathematical view point, in ordinary subdivisions, basis functions are attached to vertices. In dual subdivisions, basis functions are attached to faces (equations of faces). Moreover, in line subdivisions [10], basis functions are attached to edges (the Klein image of edges [15, 8]). So, dual subdivision is an important element of subdivisions.

2 ORDINARY SUBDIVISION

In this section, we review a general subdivision.

2.1 Subdivision Matrix

A subdivision scheme is defined by subdivision matrices and a rule of connectivity change. The subdivision scheme, when it is applied to 2-manifold irregular meshes, generates smooth surfaces at the limit. Fig. 1 is an example of the Loop subdivision. In this figure, (a) is an original mesh; subdividing (a), we get (b); subdividing (b) once more, we get (c); subdividing infinite times, we get the smooth surface (d). We call (d) the subdivision surface. Here, a face is divided into four new faces. This is a change of connectivity. In this paper, the change of connectivity is fixed to this type, but other types of connectivity change can be argued similarly.

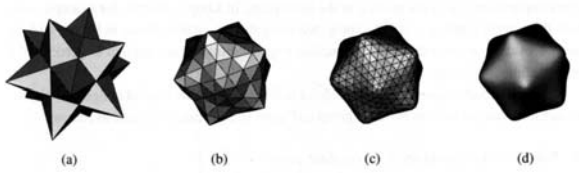


Figure 1: Loop subdivision [21].

Next, let us consider how to change the positions of the old vertices, and how to decide the positions of the new vertices. They are specified by matrices called "subdivision matrices". The subdivision matrices are defined at vertices and they depend on degree k of the vertex (the degree is the number of edges connected to the vertex). For example, Fig. 2 denotes a vertex v_0^j which has five edges. Let $v_1^j, v_2^j, \cdots, v_5^j$ be the vertices at the other terminal of the five edges. Then, subdivision matrix S_5 is defined as follows:

$$\begin{pmatrix} v_0^{j+1} \\ v_1^{j+1} \\ \vdots \\ v_5^{j+1} \end{pmatrix} = S_5^j \begin{pmatrix} v_0^j \\ v_1^j \\ \vdots \\ v_5^j \end{pmatrix}.$$

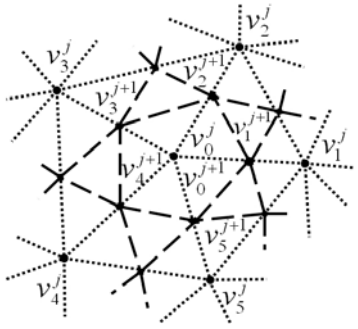


Figure 2: subdivision matrix.

Here, the subdivision matrix S_5^j is a square matrix. j means j-th step of the subdivision. Here, neighbor vertices of a vertex v are called vertices on the 1-disc of v. The subdivision matrix is generally defined not only on vertices in the 1-disc, but also on other vertices in 2-disc, 3-disc, $\cdots$. Here, we argue only subdivision matrices that affect vertices in the 1-disc. However, we can argue other subdivision matrices, similarly. In this paper, we assume that the subdivision matrix is independent on j. A subdivision scheme of this type is called "stationary".

In this way, subdivision matrix S_k is written for a vertex. However, since a newly generated vertex is computed by two subdivision matrices at the ends of the edge, the two subdivision matrices must generate the same location of the vertex. So, the subdivision matrices have this kind of restriction.

The degree k of a vertex is at least two. A vertex whose degree is two is a boundary vertex. The degree of a vertex of 2-manifold meshes is at least three. In this paper, we do not argue boundaries of meshes. So, we assume that the degree is at least three.

As seen above, a stationary subdivision scheme is defined by subdivision matrices S_k ($k \geq 3$). Then, from the theorem 2.1 in [1], the regular limit surface of subdivision $f : |K| \to \mathbf{R}^3$ is the following parametric surface:

$$f[p](y) = \sum_i v_i \phi_i(y),$$

$$v_i \in \mathbf{R}^3,\ \phi_i(y) \in \mathbf{R},\ y \in |K|, p = (v_0, v_1, \cdots)^\top,$$

where K is a complex, $|K|$ is a topological space, that is, the mesh, y is a local two-parameter, that is, locally $y = (y_1, y_2)$, i is an index of a vertex, v_i is the position of the i-th vertex, $\phi_i(y)$ is the weight function with the i-th vertex. Moreover, the weight function $\phi_i(y)$ is dependent only on the subdivision matrices. In what follows we assume that the sum of element in each row of the subdivision matrix is equal to 1. Here, the operation for generating the vertices of the $j+1$-st step of the subdivision from the vertices of the j-th step is affine invariant; it does not depend on the origin of the coordinate system.

Here, we denote $\phi(y) = (\phi_0(y), \phi_1(y), \cdots)$. Then, $\phi(y)$ decides a set of representable surfaces. Then, the set is spanned by $\phi(y)$. So, we call the weight functions basis functions. The limit surface of the subdivision is a point in such a functional space.

3 DUAL SUBDIVISION

Here, we propose a dual subdivision method.

3.1 Dual Transformation

The transformation $(p_x, p_y, p_z, p_w) \leftrightarrow p_x x + p_y y + p_z z - p_w w = 0$ is a well-known duality called projective duality in P^3. In this paper, we use $(a, b, c) \leftrightarrow ax + by + cz - 1 = 0$ which is a projective duality. We denote this duality by D, that is, for a point p, $D(p)$ represents its dual plane, and for a plane h, $D(h)$ represents its dual point. Here, the transform satisfies the following properties:

- When a point p is on a hyperplane h, and only then, the point $D(h)$ is on the hyperplane $D(p)$.
- When a point p exists in the upper (lower) half-space partitioned by a hyperplane h, the point $D(h)$ exists in the upper (lower) half-space partitioned by the hyperplane $D(p)$. Here, a lower half-space means the half-space which has the origin and the upper half-space means the other (the half-space does not contain the separating plane.).

3.2 Definition of Dual Subdivision

The ordinary subdivision is specified by how the vertices are generated and located. On the other hand, the dual subdivision, which we will define here, is specified by how the faces are generated and located.

We assumed that the sum of each row of the subdivision matrix is 1.

Here, p^j is a column vector of vertices at the j-th subdivision step. Using a subdivision matrix S, p^{j+1} is written as:

$$p^{j+1} = Sp^j,$$

where

$$p^j = \begin{pmatrix} p^j_{0x} & p^j_{0y} & p^j_{0z} \\ p^j_{1x} & p^j_{1y} & p^j_{1z} \\ & \vdots & \end{pmatrix}.$$

Therefore, if we denote

$$f^j = \begin{pmatrix} p^j_{0x} & p^j_{0y} & p^j_{0z} & -1 \\ p^j_{1x} & p^j_{1y} & p^j_{1z} & -1 \\ & \vdots & & \end{pmatrix},$$

we get

$$f^{j+1} = Sf^j,$$

where the elements of each row of f^j are coefficients of the equation $p^j_{ix} x + p^j_{iy} y + p^j_{iz} z - 1 = 0$. Therefore, the equations of planes are subdivided. These equations are the dual of vertices $(p^j_{ix}, p^j_{iy}, p^j_{iz})$. So, this subdivision is a dual framework of ordinary subdivision. Moreover, dual subdivision can be defined by a projective duality $(p_x, p_y, p_z, p_w) \leftrightarrow p_x x + p_y y + p_z z - p_w w = 0$.

Now, for any triangular mesh M, using the duality, we get a dual mesh $D(M)$. Here, if the degree of a vertex v of M is k, then v is the intersection of faces $f_i,\ i = 1, 2, \cdots, k$, so these vertices $D(f_i),\ i = 1, 2, \cdots, k$ is on the face $D(v)$. So, we get following observation:

Observation 3.1 (Dual mesh)
If the degree of the vertex v of M is k, the face $D(v)$ of $D(M)$ is a k-gon.

Note that even if meshes in primal space are bounded, dual meshes are not necessarily bounded. First, clearly, dual meshes depend on the origin in primal space. A point $\alpha a + \beta b$ on a face p is the convex combination of neighbor vertices a, b. So, we can see the plane $D(\alpha a + \beta b) = \alpha D(a) + \beta D(b)$ is a tangent plane of $D(p)$. Thus we can define the tangent plane of $D(p)$ as convex combinations of tangent planes of the neighborhood (See the upper figure in Fig. 3.). If a tangent plane $\alpha p + \beta q$ of a vertex contains the origin in primal space, then the vertex $D(\alpha p + \beta q)$ is a point at infinity, and so the face, which has vertices $D(p), D(q)$, is not bounded (See the lower figure in Fig. 3.). Therefore, we want to know the condition for dual meshes and dual subdivision surfaces to be bounded. So, we derived a necessary and sufficient condition for dual meshes and a sufficient condition for dual subdivision surfaces in [9].

Here, we define the rule of connectivity change of dual subdivision. The connectivity change of dual subdivision is defined as dual of the connectivity change of ordinary subdivision (See Fig. 4).

So, we get following observation.

Observation 3.2 (Dual subdivision)
Applying ordinary subdivision to meshes in the primal space is dual of applying dual subdivision to dual meshes in the dual space.

Here, we can see that if meshes made by ordinary subdivision approximate surfaces very well, dual meshes made by dual subdivision approximate dual surfaces very well, too. Thus, dual subdivision has a useful property that it can represent surfaces by "flat" polygons on non-triangular meshes, for example, hexagonal meshes.

3.3 Properties of Duality

In this subsection, we explain an important property. The property is based on dual transformation $(a, b, c) \leftrightarrow ax + by + cz - 1 = 0$ and 2-manifold.

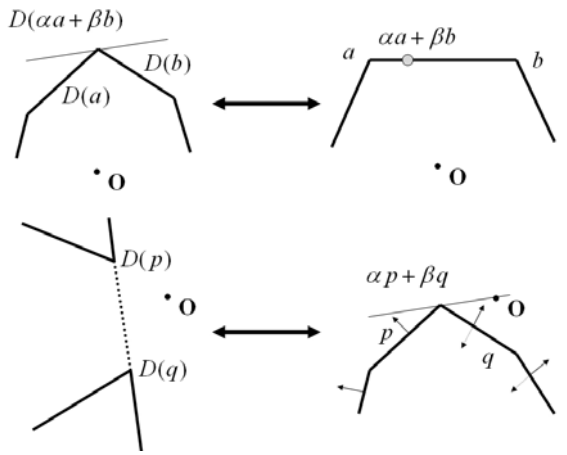


Figure 3: Tangent planes of a vertex. In the upper figure, we can see the tangent plane of a vertex as convex combinations of tangent planes of the neighborhood. So, if a tangent plane $\alpha p + \beta q$ contains the origin, $D(\alpha p + \beta q)$ is a point at infinity. Therefore, the face is not bounded. Here, q is a plane $-p_x x - p_y y - p_z z + 1 = 0$ whose normal is $(-p_x, -p_y, -p_z)$. If we write the equation of q as $p_x x + p_y y + p_z z - 1 = 0$, then the normal is (p_x, p_y, p_z) (this normal is denoted by the dashed arrow). We consider that meshes have continuous normals, that is, meshes are oriented. Thus, we define the equation of q as $-p_x x - p_y y - p_z z + 1 = 0$. Then, the un-bounded face is convex combinations of $D(p)$ and $D(q)$, too.

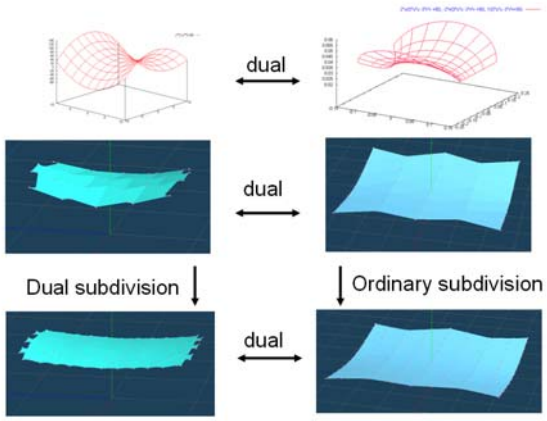


Figure 4: The duality between ordinary subdivision and dual subdivision. Upper left drawing is a surface which has saddle points. Upper right drawing is the dual surface. Middle right picture is a triangular mesh made by plotting points on upper right surface. Middle left mesh is the dual mesh of the middle right mesh. we get lower right mesh made by ordinary subdivision for the middle right mesh. On the other hand, we get lower left mesh made by dual subdivision for the middle left mesh. Then, the lower left mesh is the dual mesh of the lower right mesh. Dual subdivision is defined as such. Like this, dual subdivision can represent surfaces which have saddle points.

So, the property holds independently of the dual subdivision.

Here, we denote a regular 2-manifold in the primal space as S, the dual shape of S as $D(S)$. Points of $D(S)$ are the dual of tangent planes of S. Tangent planes of $D(S)$ is the dual of points of S. We can see $D(S)$ as an envelope surface defined by the dual of points of S. Here, "flat" means that the points of the subset of a surface share a tangent plane.

In this paper, we assume that a 2-manifold is connected and has a finite genus.

First, we get following proposition.

Proposition 3.1 (Duality of 2-manifold)
If S is a bounded and regular 2-manifold, and if any subset of S and $D(S)$ are not flat, then $D(S)$ is a 2-manifold.

Proof *First, any bounded and regular 2-manifold S in the primal space has an open covering which is composed of a finite number of open sets whose topology is equal to that of a disc. Moreover, this dual transform is a continuous and one-to-one mapping. So, the open sets are mapped to open sets of tangent planes in the dual space. Since any subset of S is not flat, an envelope surface made by an open set of tangent planes is an open set of points in the dual space. Therefore, the dual surface of the open covering is represented by a union of open sets. Since the open covering is composed of a finite number of open sets, the dual surface of the open covering is an open set. Moreover, there is an open covering whose union is equal to S. So, the dual surface of this open covering is equal to $D(S)$. Therefore, there are open sets, whose topology is equal to that of a disc, at any point of $D(S)$. So, $D(S)$ is a 2-manifold.* □

$D(S)$ is generally not a 2-manifold even if S is a 2-manifold. For example, if S is a plane, then $D(S)$ is a point, because a subset of S is flat. For example, a plane is totally flat, a face of a mesh is flat, the top and bottom circle of a torus are flat. So, flat parts make degenerations for dual shapes. If the degeneration does not break the structure of 2-manifold of dual surfaces, we can easily make the degenerated surface. Otherwise, we must use special subdivision matrices and a rule of connectivity change to represent the degenerated surface.

Therefore, in this paper, we discuss non-degenerate surfaces.

4 INFLECTION PLANE

4.1 Inflection Point

We will discuss the smoothness of the limit surfaces of dual subdivision and show the duality of smoothness $C^1_{\mathbf{ordinary}} \Leftrightarrow C^1_{\mathbf{dual}}$ in P^3. However, to show the relation, we need a condition. In this section, we talk about the condition.

Even if a dual surface in the dual space is smooth and if any subset of the surface is not flat, the associated surface in the primal space is not necessarily smooth. (However, the continuity of tangent planes is guaranteed.) For example, see Fig. 5.

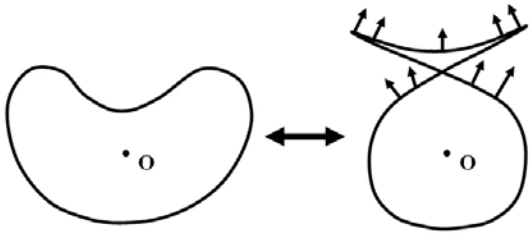


Figure 5: The left object in 2D has inflection points. The right object, which is the dual of the left object, has reversals of the normal at dual inflection points. In 3D, such thing happens, too.

Although the dual curve in the dual space is smooth, we see that the corresponding curve is not smooth in the primal space. This arises from reversals of normals at the dual of inflection points.

Here, we define an "inflection point" of surfaces in $\mathbf{R}^3$. We call a point p an "inflection point" if a cross-sectional curve of the surface at p (the cross-sectional curve is the intersection of the surface and a plane through p and the origin.) has the inflection point p.

Thus, if the C^1 dual surface has inflection points, the associated surface in the primal space is not C^1-continuous.

Consider the neighborhood of a point. We can classify the neighborhood to convex, concave, inflection. As seen above, inflection parts break the smoothness. On the other hand, the others do not break it.

4.2 Inflection Plane

To overcome this problem, we use a technique named "inflection plane".

First, we get a mesh in the primal space as Fig. 6. Applying smooth subdivision scheme to this mesh, the limit curve in primal space is smooth. So, the limit curve is not equal to the target shape in Fig. 6. Therefore, dual of the limit curve of this mesh is not equal to the left shape in Fig. 5.

Even if the original mesh in the primal space approximates the right shape very well, the limit curve in the primal space does not have reversals of normals. So, the curve is not equal to the target shape.

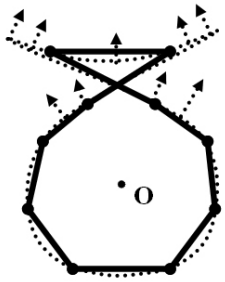


Figure 6: A mesh in primal space.

So, we want to generate the surface with reversals of normals by subdivision in the primal space.

Here, we define an "inflection plane". We add two-ply faces $AB, A'B, \cdots$ as shown in Fig. 7. These added faces generate the reversals of normals, because the basis function at point B is the delta function (Here, we assume all supports of basis functions are 1-disc. Then, the limit curve of the mesh to which $AB, A'B$ is added is tangent plane continuous.).

Like this, the two-ply face $AB, A'B$ generates the dual of the tangent plane at an inflection point in the dual space. Moreover, the tangent plane at B is the dual of an inflection point in the dual space. So, we call the two-ply face $AB, A'B$ an "inflection plane".

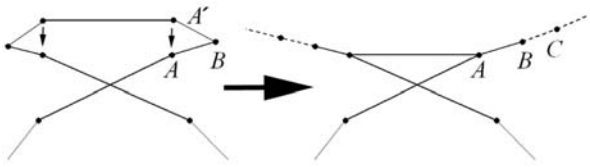


Figure 7: Adding an inflection plane $AB, A'B$. We conform the position of vertex A' to that of vertex A. So, both meshes have the same topology. We call the two-ply face $AB, A'B$ "inflection plane".

If all supports of those are over 2-disc, we add points $C, D, \cdots$. Then, we get the limit surfaces with tangent plane continuity and reversals of normals.

Similarly, using inflection planes, we can get smooth surfaces which have inflection points in dual space (see Fig. 8).

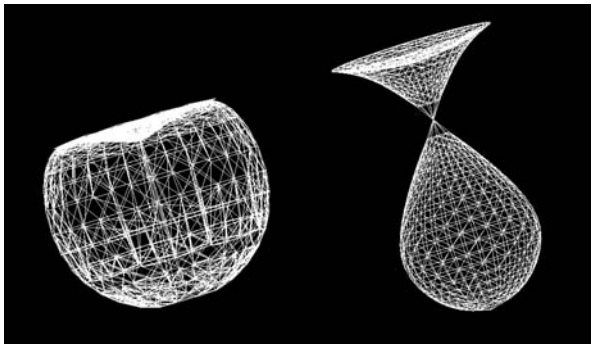


Figure 8: The left mesh which is made by dual subdivision is the dual of the right mesh and not convex and has inflection points. The right mesh which is made by ordinary subdivision using inflection plane has dual inflection points.

5 DUALITY OF SMOOTHNESS

In this section, we derive the relation $C^1_{\mathbf{ordinary}} \Leftrightarrow C^1_{\mathbf{dual}}$ in P^3. Then, we can get smooth dual limit surfaces.

Proposition 5.1 (Duality of smoothness) *Assume that S is a bounded and regular 2-manifold and any subset of S and $D(S)$ are not flat. Then $D(S)$ is tangent plane continuous if and only if S is tangent plane continuous. Moreover, if S and $D(S)$ have no inflection points, then $D(S)$ being C^1-continuous is equivalent to S being C^1-continuous.*

Proof *Since S is tangent plane continuous, points of $D(S)$ are continuous. Moreover, any subset of S is not flat. So, the dual of tangent planes of S is non-degenerate. Here, points of S are continuous. So, tangent planes of $D(S)$ are continuous. Therefore, $D(S)$ is tangent plane continuous. Similarly, since $D(S)$ is tangent plane continuous, points of S are continuous. Moreover, any subset of $D(S)$ is not flat. So, the dual of tangent planes of $D(S)$ is non-degenerate. Here, points of $D(S)$ are continuous. So, tangent planes of S are continuous. Therefore, S is tangent plane continuous.*

If S and $D(S)$ do not have inflection point, then $D(S)$ is C^1-continuous if S is C^1-continuous. Because, locally the neighborhood of a point of S is convex or concave. So, the neighborhood does not reverse normals of $D(S)$. □

To get smooth $D(S)$ which has inflection points, we must use the inflection plane.

5.1 Properties of Dual Subdivision

Next, we derive an important theorem for smoothness of limit surfaces.

Theorem 5.1 (Duality of smoothness) *Assume that there are local parameterizations, which have Jacobi matrix of maximal rank 2 except at extraordinary points, on basis functions of ordinary stationary subdivision, and there are unique tangent planes at extraordinary points, and any subset of the limit surface is not flat, and the limit surfaces of ordinary and dual subdivision have no inflection points. Then, the limit surfaces of the dual subdivision are C^1-continuous if and only if the limit surfaces of the ordinary subdivision are C^1-continuous:*

$$C^1_{\mathbf{ordinary}} \Leftrightarrow C^1_{\mathbf{dual}}.$$

Proof *For any basis function generated by ordinary stationary subdivision, if Jacobi matrix is degenerate at a point on the basis function except the extraordinary point, then Jacobi matrix is degenerate at any point of the basis function. Then, Jacobi matrix is degenerate on the part, which corresponds to the basis function, of the subdivision surface except a finite number of extraordinary points. So, Jacobi matrix of maximal rank 2 except at extraordinary points means that Jacobi matrix is non-degenerate at any point of the surface generated by ordinary subdivision except extraordinary points. Therefore there are unique tangent planes of the limit surface of ordinary subdivision except extraordinary points. Here, there are unique tangent planes at extraordinary points. So, there are unique tangent planes of the surface generated by ordinary subdivision. Thus, any subset of $D(S)$ is not flat. Moreover, any subset of the limit surface is not flat. Therefore, by proposition 5.1, we get this theorem.* □

In this way, we can guarantee the smoothness of dual limit subdivision. So, we can use the dual subdivision scheme for applications, e.g. approximating shapes.

6 CONCLUSION

In this paper, we proposed a new subdivision method. This is a dual framework of ordinary subdivision based on the projective duality. Because of the duality, dual subdivision has useful properties similar to ordinary subdivision.

First, we derived the duality of 2-manifold. Thus, we can see that the dual limit surface is 2-manifold.

Second, we defined an “inflection plane". Using the inflection plane, we can represent smooth surfaces with inflection points by dual subdivision.

Finally, we derived the relation $C^1_{\mathbf{ordinary}} \Leftrightarrow C^1_{\mathbf{dual}}$. This duality of smoothness enables us to represent smooth surfaces by “flat" non-trigonal polygons.

Moreover, we can lead multiresolution analysis [21] for dual subdivision. It is useful for the mesh editing, watermarking, etc..

For higher-order smoothness, refer to our technical report [9]. In the technical report, we derived conditions for the limit surface of dual subdivision surfaces to be C^k-continuous. Using "universal surface" [24], we derived relations of smoothness between ordinary subdivision and dual subdivision.

REFERENCES

[1] A. S. Cavaretta, W. Dahmen, and C. A. Micchell. Stationary subdivision. *Memoirs Amer. Math. Soc.*, 93(453), 1991.

[2] Johan Claes, Koen Beets, and Frank Van Reeth. A corner-cutting scheme for hexagonal subdivision surfaces. In *Proceedings of Sape Modeling International 2002*, pages 13–24. IEEE, 2002.

[3] T. Derose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *SIGGRAPH '98 Proceedings*, pages 85–94. ACM, 1998.

[4] D. Doo and M. A. Sabin. Behaviour of recursive subdivision surfaces near extraordinary points. *Computer Aided Geometric Design*, 10:356–360, 1978.

[5] Matthias Eck, Tony DeRose, and Tom Duchamp. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95 Proceedings*, pages 173–182. ACM, 1995.

[6] G. Farin, J. Hoschek, and MS Kim. *Handbook of Computer Aided Geometric Design*. Elesvier Science Publishers, 2002.

[7] T. N. T. Goodman, C. A. Micchell, and W. J. D. Spectral radius formulas for subdivision operators. In *L. L. Schumaker and G. Webb, editors, Recent Advances in Wavelet Analysis*, pages 335–360. Academic Press, 1994.

[8] Bert Jutter and Katharina Rittenschober. Using line congruences for parameterizing special algebraic surfaces. In *the Tenth IMA International Conference on the Mathematics of Surfaces*, Leeds, 2003.

[9] Hiroshi Kawaharada and Kokichi Sugihara. Dual subdivision a new class of subdivision schemes using projective duality. METR 2005-01, The University of Tokyo, January 2005.

[10] Hiroshi Kawaharada and Kokichi Sugihara. Line subdivision. In *the Eleventh IMA International Conference on the Mathematics of Surfaces*, Loughborough, 2005.

[11] Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surface. In *SIGGRAPH '00 Proceedings*, pages 85–94. ACM, 2000.

[12] David Levin and Ilana Wartenberg. Convexity-preserving interpolation by dual subdivisions schemes. In *Curve and Surfaces*, St. Malo, 1999.

[13] C. T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[14] Michael Lounsbery, Tony Derose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.

[15] Helmut Pottman and Johannes Wallner. *Computational Line Geometry*. Springer, 2001.

[16] H. Prautzsch. Analysis of c^k-subdivision surfaces at extraordinary points. *Preprint. Presented at Oberwolfach*, June 1995.

[17] U. Reif. A unified approach to subdivision algorithms near extraordinary points. *Computer Aided Geometric Design*, 12:153–174, 1995.

[18] Ulrich Reif. A degree estimate for polynomial subdivision surfaces of higher regularity. *Proc. Amer. Math. Soc.*, 124:2167–2174, 1996.

[19] Lluis Ros, Kokichi Sugihara, and Federico Thomas. Towards shape representation using trihedral mesh projections. *The Visual Computer*, 19:139–150, 2003.

[20] Jos Stam. Evaluation of loop subdivision surfaces. In *SIGGRAPH 98 Conference Proceedings on CDROM*. ACM, 1998.

[21] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers, 1996.

[22] Joe Warren and Henrik Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann Publishers, 1995.

[23] D. Zorin. *Subdivision and Multiresolution Surface Representations*. PhD thesis, University of California Institute of Technology, 1997.

[24] Denis Zorin. Smoothness of stationary subdivision on irregular meshes. *Constructive Approximation*, 16(3):359–397, 2000.

A Declarative System to Design Preliminary Surfaces

Raphaël La Greca
raphael.la.greca@esil.univ-mrs.fr

Marc Daniel
marc.daniel@esil.univ-mrs.fr

LSIS (UMR CNRS 6168)
Case 925 - 163 Av. de Luminy
13288 Marseille cedex 09 - (France)

ABSTRACT

B-Spline and NURBS surfaces are most often considered to model objects. The object shape is designed by manipulating several control points, which is often very complex and tedious. The declarative approach of surface modelling is a fast and easy way to obtain sketches of parametric surfaces. The designer provides a description of the shape he/she wants to obtain. The semantic extracted from this description is structured through XML language. As a result, a set of parametric surfaces corresponding to the given constraints and features is proposed to the user. This approach is specially devoted to speed up the preliminary design process. This paper introduces our system as a high level tool of surface modelling. Details dealing with the different models and processing involved in our system are proposed. The document is illustrated by the first results of our research study.

Keywords

Geometric modelling, Declarative, Surface, NURBS, semantics, CAD, Geometric constraints.

1. INTRODUCTION

Present days, Computer Aided Geometric Design is a key computing field in industrial activities. This technology enables to visualise objects and to simulate their behaviours before being manufactured. In most instances, these objects are designed by a set of NURBS surfaces. Their shape can be manipulated by several control points which can often be very complex and tedious. In order to help the designers and to provide them a high level tool of design, we suggest a declarative approach during the modelling process. The purpose of this system is to produce easily and quickly sketches of NURBS surfaces [Dan96]. The designer has only to give to the declarative system a description of the shape he/she wants to obtain, and the process suggests him/her one or several solution models which satisfy the specified constraints and features [CDMM97]. This relieves the user of any knowledge about the underlying mathematical model and the designer can really express his/her creative feeling. He/she can see all the surfaces which correspond to the description. If they match the requirements the process can be stopped. Otherwise, he/she can change or improve the description and start again the declarative process to receive new solutions. This loop can be iterated until an expected shape is produced. This paper presents the framework of the declarative system of surface modelling we developed (see Figure 1).

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

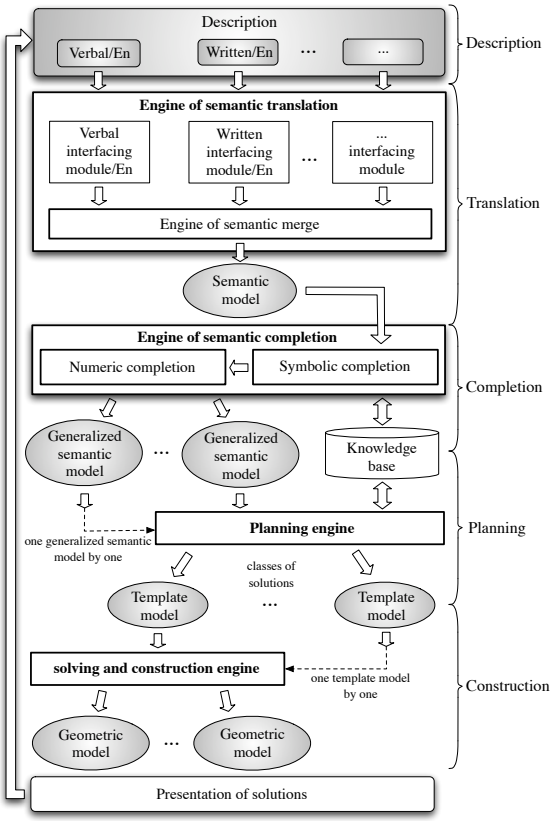


Figure 1: The framework of the declarative system.

The first part introduces the description concept and its translation. The three next sections give details about

the three other steps, Completion, Planning and Construction, used to obtain one or several solution surfaces. Some practical results and future works conclude this document.

2. THE DESCRIPTION AND ITS SEMANTIC TRANSLATION

The input of the declarative system we developed is a description of the surface expected by the designer. The purpose of the translation process is to extract the semantic from the description, and to structure it into a single final model called *semantic model*.

2.1 From the Description To the Semantic

A declarative system must be able to adapt itself to the user. In order to follow this rule, we introduce the concept of interfacing modules. A description can be given by the designer using several media like written English, verbal French or dialog box and button interface. In every instance, the description depends of the situation and the knowledge of the designer. Thus, an object can be described differently by many people. We choose to focus our work on the mechanical domain to obtain trade-oriented descriptions. A specific vocabulary is identified and can be classified into eight categories: comparing, junction, constraint, topology, morphology, quantification, localisation and geometry. Each term is a key word with a signification. One interfacing module is dedicated to translate one specific media into a structured semantic language by key words extraction and knowledge organisation. All the knowledge extracted from the description is stored in the semantic model using a XML[1] tree. The XML format has been chosen because it allows us to easily structure the description and because it is well-adapted to introduce specific fields which model the semantic of the final surfaces. Let us propose a very simple example based on a written English description:

> *" The surface has a rectangular shape and its right part is a little bulged. "*

After the semantic translation step, the corresponding semantic model could be:

```
<Surface Id="Z0">
   <Shape Id="RECTANGULAR"/>
   <Zone Id="Z1">
      <Deformation Id="BULGED">
         <Quantifier Id="LITTLE"/>
      </Deformation>
      <Localisation Id="RIGHT"/>
   </Zone>
</Surface>
```

This XML representation is a direct translation of the designer description. In this case of written text, the data processing of the corresponding interfacing module consists of parsing the description sentence to identify key words [EH04] and to fill out the matching XML fields. An interfacing module is considered like a plugin which can be switched on or off according to the requirements of the situation.
If several media are used to describe the same object or surface, all the structured semantics is merged by the *engine of semantic merge* in order to obtain a single semantic model. This last model is a semantic representation of the surface expected by the designer.

2.2 Semantic Model Structure

The XML structure of a semantic model is organised around the notion of zones also called parametric zones. A surface is viewed as a set of zones. Each of them can be a child of another one and is defined by a localisation, a shape, a set of deformations, a set of constraints and a closed polygon inferred from all these data. This framework is illustrated by Figure 2 using the UML notation.

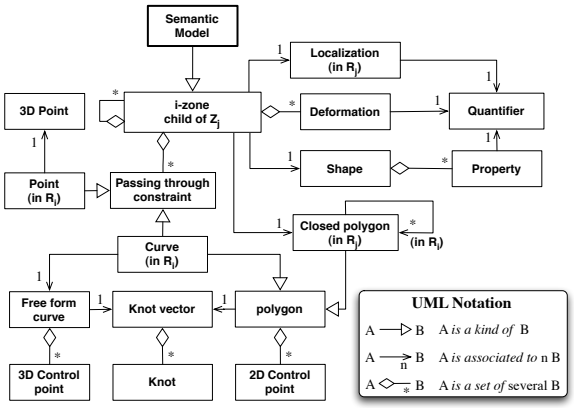


Figure 2: Framework of a semantic model using the UML notation.

Thus, a declarative surface is defined as the root of the XML tree of which each node is a zone. We can formalise this framework with the following definition:

Definition 2.1 - Let Z_i, child of Z_j, be the *i-zone* defined by the set $\{Z,S,L,P,D,C\}$ where:

- Z is the set of all local zones defined on Z_i. If Z is empty, Z_i has no child, otherwise a child-zone is valid only if its shape is fully included in the shape of Z_i. The bounding box of Z_i is written R_i and defines the new coordinate system of all its child-zones (see Figure 3). We choose to fit the area of each new coordinate system to $[0,100] \times [0,100]$. It is very important to remark that R_0 which is the coordinate system defined by the root-zone, will be matched with the parametric plane of each solution surface during the construction process. A child-zone is defined with the `<Zone>...</Zone>` markups. The interweaving of all these sets Z leads to the tree framework of the XML structure. This tree is called structuring tree and is denoted by T_s,

[1] e**X**tended **M**arkup **L**anguage.

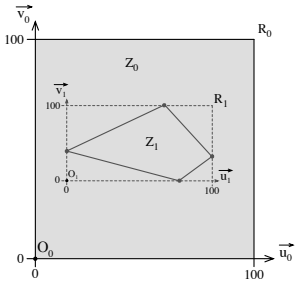


Figure 3: Reference frame of the declarative surface.

- S is the symbolic data which describes the shape of Z_i. This data must be in the knowledge base and could be, for example, "QUADRATE" or "ROUND". The `<Shape>...</Shape>` markups are used to store this data on the XML structure,
- L is the symbolic data which describes the localisation of Z_i. Two kind of localisation can be distinguished, the relative one and the absolute one. The first can locate Z_i according to other objects or zones, the second can directly situate Z_i on R_j. This data must be in the knowledge base. For example, "ON THE RIGHT OF" or "NEAR TO" could be used in the relative localisation case and "RIGHT PART" or "TOP LEFT CORNER" in the absolute localisation case. The `<Localisation>...</Localisation>` markups are used to store this data on the XML structure. The difference between the two localisations is made using the `<Relative>...</Relative>` and `<Absolute>...</Absolute>` markups,
- P is the closed control polygon of the NURBS curve C_i defining the borderline of Z_i on R_j. If the degree of C_i is equal to 1 the borderline is exactly P. If P is not directly given by the designer, P is inferred during the numeric completion process, using the knowledge base and the symbolic data of Z_i which describe its shape and its localisation on R_j. It is possible to define some areas on Z_i as holes in order to obtain zones of different topology (see Figure 4),

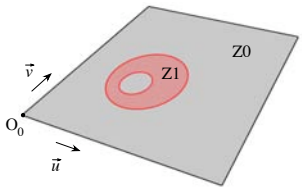


Figure 4: $Z1$, a zone with a hole.

  The `<Polygon>...</Polygon>` markups are used to store all the polygon data. The markups `<Minus>...</Minus>` add the holes definition of P on the XML structure,

- D is a set of deformations to apply on Z_i. A deformation is also called a *soft constraint* which is the final aspect the designer expects to find on the solution surfaces. Each deformation can be linked to a quantifier which is able to control its magnitude. The available deformations and their relationships are stored on the knowledge base. The `<Deformation>...</Deformation>` markups are used on the XML structure to list the deformations to be applied on Z_i,
- C is a set of constraints or *strict constraints* which have to exactly be satisfied on the solution surfaces. One of these constraints is typically to pass through one or many specific points of the space. These constraints are stored on the XML structure by the `<Constraints>...</Constraints>` markups. To add a new constraint, we use specific markups: `<Point>...</Point>`.

This XML structure must be complete to be exploited during the planning process. That is why the completion process has to add the missing data according to the knowledge base of our system.

3. SEMANTIC COMPLETION

The input of this process is a semantic model. Its purpose is to check if the XML structure of this model is complete. In such a case, the semantic model is called *generalised semantic model*. Otherwise the completion process has to add the missing data using the *knowledge base* to generate one or several generalised semantic models. This processing is divided into two parts: the *symbolic completion* and the *numeric completion*.

3.1 The Knowledge Base

The knowledge base stores all the available data used over all the processing steps. The information is organised using the XML language. We can find the definition of all the available shapes of zone. Their borderlines are defined by NURBS curves to be able to obtain several different rounded shapes.

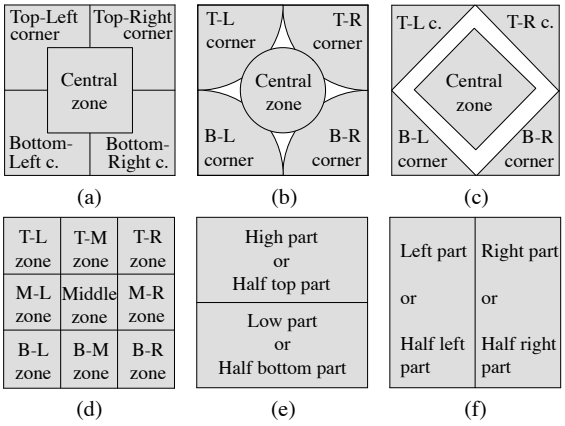
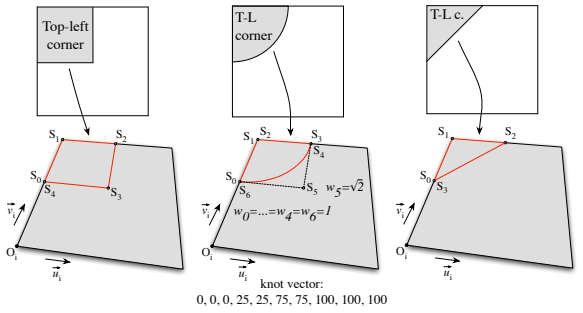


Figure 5: Some examples of absolute localisation.

All the available absolute localisations (see Figure 5) are also stored on the knowledge base. For one localisation, it is possible to find several different *settings* like the `Top-Left corner` illustrated in Figures 5.a, 5.b and 5.c. Each available setting is defined in the knowledge base by a specific closed NURBS curve (see Figure 6).



(a) Square setting (1-degree) (b) Round setting (2-degree) (c) Triangular setting (1-degree)

Figure 6: NURBS definitions of different settings of the `Top-Left corner` localisation.

The knowledge base contains the *graph of dependences between deformations* written G_{dbd}. This graph stores the order of deformations to apply if many of them have to be performed on a same area of a surface. The nodes of G_{dbd} are the available deformations of our system. A single oriented link from a deformation *A* to a deformation *B* means that *A* must be applied before *B*. A double oriented link means that two cases must be considerated, the case when *A* is applied before *B* and the case when *B* is applied before *A*. A part of the G_{dbd} graph is shown in Figure 7. The deformation abbreviations used are:

Ar:	Arched	*Bl*:	Bulged	*Lu*:	Lumpy
Ho:	Hollowed	*Hl*:	Hole		

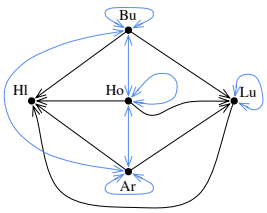


Figure 7: Part of a *graph of dependences between deformations*.

The lumpy deformation must be done at the end. We made this choice because this deformation is viewed as a finishing touch.
The knowledge base also stores all *quantifiers* the designer can use in the description like “LITTLE” or “VERY”. The goal of a quantifier is to shade properties, localisations and deformations according to a value from 0 to 100. By definition, a declarative approach must allow the user to give a fuzzy description [Des00]. In order to keep this very important aspect of the declarative design, each quantifier is assigned to a fuzzy set (see Figure 8).

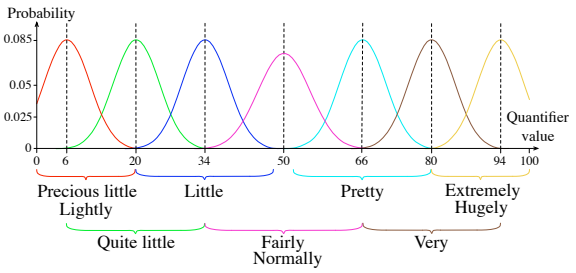


Figure 8: Fuzzy sets assigned to each quantifier.

Each fuzzy set is here defined by a Gaussian probability density function. Using all these data stored in the knowledge base, the completion process begins by the *symbolic completion*.

3.2 Symbolic Completion

In most cases, the designer gives to the system an incomplete description. This can happen because he/she forgets an information or because the data seem to be obvious for him/her. Nevertheless, it is not so obvious for the system. Thus, the latter has to find and complete all the missing data of the current semantic model. The technique is iterative. Each iteration consists in applying several simple rules on existing data to infer the missing ones. This is for example two rules to enforce:

- *If a zone has no deformation, the deformation “NONE” is added to this zone,*
- *If a deformation has no quantifier, the quantifier “FAIRLY” is added to this deformation.*

Some other rules entail the generation of several more precise semantic models. For example, if the designer does not give the shape of a zone or does not specify the localisation setting to consider, the system must generate a new semantic model for each possible shape or setting. The symbolic completion is carried on using all these new models. The process is stopped when no rule can anymore be applied. At the end of the symbolic completion, several semantic models are available and the numeric completion is applied on each of them.

3.3 Numeric Completion

This step adds to each semantic model the numeric data required to be processed. The method is not iterative. Each model is processed one by one using several simple rules, like for example:

- *If a quantifier has no specified values, the corresponding data are added from the knowledge base,*

- *If the control polygon of a zone is not present, all its control points are placed on the corresponding local reference frame according to the localisation and the shape of this zone.*

The second rule is one of the most important because the control polygons define the zone borderlines which are principally used during the *planning process* and the *construction process*. After the numeric completion, each semantic model is complete and represents the knowledge of the final shapes the system will produce. These full models are called *generalised semantic models* and each of them will be taken into consideration by the two last steps.

4. CONSTRUCTION PLANNING

A generalised semantic model is made of zones which are linked to one or many deformations. If many zones overlap each other, their deformations have to be applied on the shared area. The deformations are usually not commutative as illustrated in Figure 9.

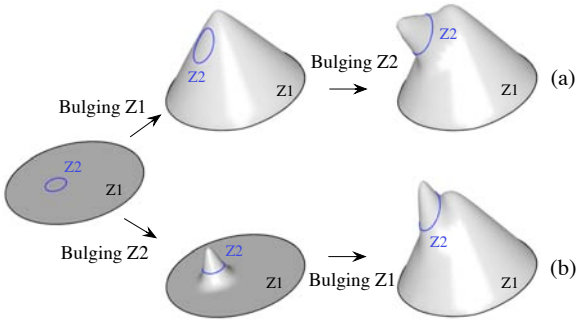


Figure 9: Non-commutativity of the "bulged" deformation.

That is why, it is possible to obtain several kinds of solution according to the overlaps and the deformations of each zone. The purpose of the planning process is to generate all possible sequences of deformations in order to obtain distinct solution surfaces. This is done through the analysis of the *graph of dependences between zones* written G_{dbz} which stores the ordering constraints of application of deformations to all the zones of the surface. Each node of this graph is an operation (*zone*, *deformation*) which can be read "*Apply the deformation called deformation to the zone called zone*". If a zone has two deformations, the graph has two nodes which involve the same zone but with a different deformation to each of them. The edges of this graph are symbolised by two kinds of links, le single oriented link and the double oriented link which meanings are similar to the ones of the graph of *dependences between deformation* stored in the knowledge base (see section 3.1). Figure 10.c illustrates the *graph of dependences between zones* obtained from the structured tree of Figure 10.a and the position in R_0 of the corresponding zones which is shown by Figure 10.b. This example uses these data: *Z0 is lumpy, Z1 is arched, Z2 and Z5 are hollowed, Z3, Z4 and Z6 are bulged.*

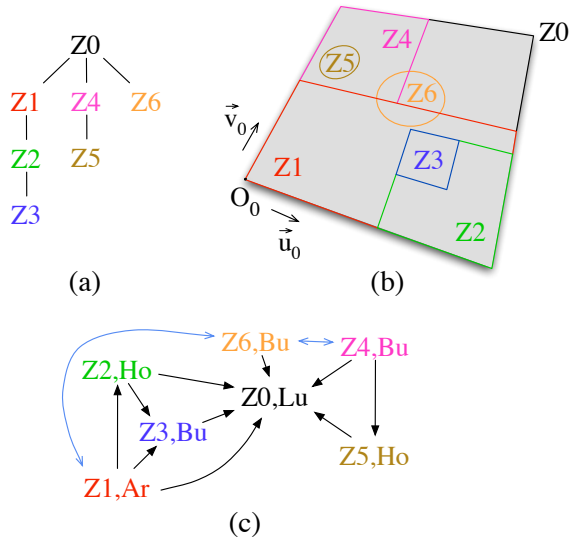


Figure 10: (a) Structuring tree, (b) possible zone layout and (c) the corresponding graph of dependences between zones.

The aim of the analysis of G_{dbz} is to find all the construction sequences of (*zone*, *deformation*) to apply in order to obtain all the different solution surfaces. A single oriented link in G_{dbz} is an ordering constraint. Thus if we have (*Z2*,*Ho*)→(*Z3*,*Bu*), the resulting sequence is (*Z2*,*Ho*)(*Z3*,*Bu*) which can be read: "First *Z2* is hollowed, next *Z3* is bulged". A double link in G_{dbz} means that two cases are possible and gives different results. Thus if we have (*Z1*,*Ar*)↔(*Z6*,*Bu*), a first class of solutions is the sequence (*Z1*,*Ar*)(*Z6*,*Bu*) and a second class of solutions is (*Z6*,*Bu*)(*Z1*,*Ar*). This last step analyses all these combinations and infers all the different classes of solutions corresponding to G_{dbz}. One class of solutions can contain several construction sequences which are equivalent. That is, if all the deformations of each construction sequence are applied, the resulting surfaces are strictly the same. One construction sequence is randomly chosen in each class of solutions to represent it. This chosen sequence comes in addition to the generalised semantic model to become a *template model*. The graph analysis of Figure 10.c generates four classes of solutions represented by these four template models:

→ `Template model 1: (Z6,Bu) (Z4,Bu) (Z5,Ho) (Z1,Ar) (Z2,Ho) (Z3,Bu) (Z0,Lu)`
→ `Template model 2: (Z4,Bu) (Z5,Ho) (Z1,Ar) (Z6,Bu) (Z2,Ho) (Z3,Bu) (Z0,Lu)`
→ `Template model 3: (Z4,Bu) (Z6,Bu) (Z1,Ar) (Z5,Ho) (Z2,Ho) (Z3,Bu) (Z0,Lu)`
→ `Template model 4: (Z1,Ar) (Z2,Ho) (Z6,Bu) (Z4,Bu) (Z5,Ho) (Z3,Bu) (Z0,Lu)`

At the end of the planning process, several template models can be generated. To present one possible solution surface per template model, the system makes an instance of each of them and shows them to the designer. The surface modelling is done by the construction process.

5. CONSTRUCTION OF SURFACES

The input of the *solving and construction engine* is one template model. This process consists in solving the soft and the strict constraints given by the designer. At the first time, our method consists in creating a surface of degree 3×3 defined by 4×4 control points which are placed on a plane and two uniform clamped knot vectors. The parametric plane of this surface and the reference frame of the root zone R_0 are merged to link the zone definitions with the surface. This is the initial surface which will be deformed to obtain one instance of the current template model. Each deformation needs control points to be performed. That is why the net of control is refined by adding lines and columns of control points inside the zones which have to be deformed. This operation is done using the knot insertion algorithm developed by Boehm [BP85]. The control points which are inside the borderlines of a zone are assigned to it in order to be moved during the deformation of this zone.

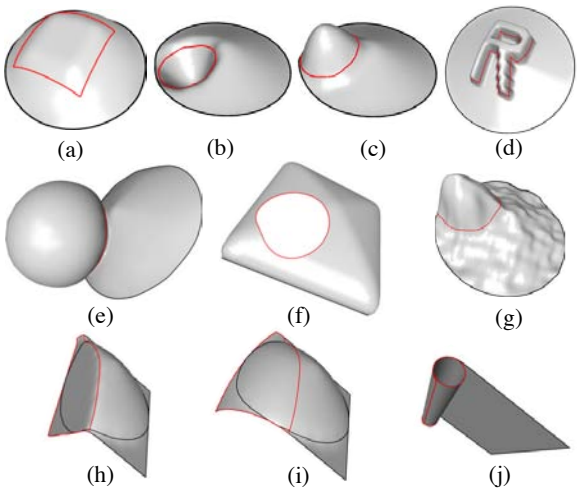


Figure 11: Available deformations: (a) $flattened$, (b) *hollowed*, (c) *bulged*, (d) *extruded*, (e) $inflated$, (f) *hole*, (g) $lumpy$, (h) $folded$, (i) *arced* and (j) *rolledup*.

In a second time, our method solves the soft constraints sculpting the global shape of the surface. This step consists in deforming the net of control of the surface defined by refinement. Each operation (*zone*,$deformation$) is done one by one applying the $deformation$ to the corresponding *zone* following the construction sequence of the current template model. Many deformation techniques [PT97, Per04] can be used. However, we choose to develop our geometric tools to be closer to our needs. The current available deformations of our declarative modeller are illustrated in Figure 11. Each deformation is set by its qualifier stored in its semantic model. Because of the fuzzy approach of the qualifiers, each launching of the current engine can generate several different[2] instances of the same template model. At the end of this construction step, the surface is a sketch of the designer expected surface. Figure 12 illustrates one instance of template 4 of the last example with its net of control.

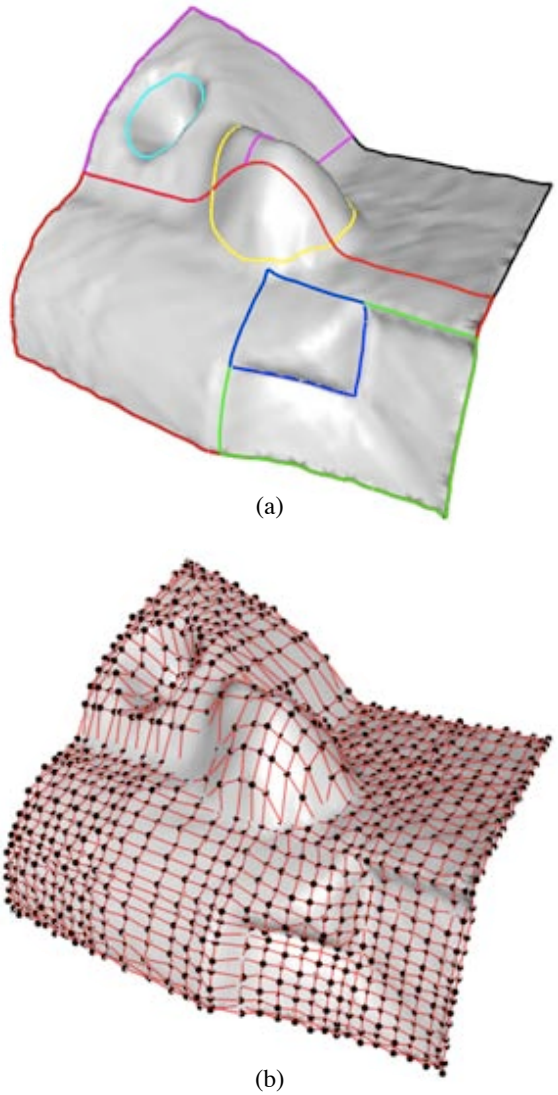
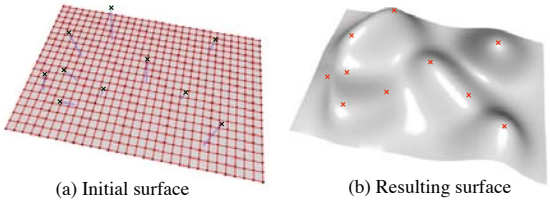


Figure 12: Template model 4 of the previous example with its zones (a) and with its net of control (b).

In a third time, the engine solves the strict constraints[3] using a geometrical solving method we developed [LDB05]. This method is able to set the range of each punctual constraint in order to control the

[2] but pretty close.
[3] passing through specific points in space.

local deformations of the surface. The method can also be applied to satisfy several constraints. This case is illustrated in Figure 13 where a constraint is symbolised by a cross.



(a) Initial surface (b) Resulting surface

Figure 13: Deformation of a B-spline surface of degree 3×3 defined by 30×20 control points to satisfy ten constraints with a medium range of influence.

At the end of the construction process, one or many instances of resulting template models are presented as solution surfaces to the designer. He/she can choose the one which is closest to his/her requests. If no surface satisfies the designer, he/she can modify the description and start again the declarative system. In order to illustrate the current capabilities of our system, the next section presents two practical results.

6. FIRST RESULTS

The two practical results introduced in this section were obtained using our first prototype of declarative modeller of surfaces. We give it a possible generalised semantic model and it proposes us one or several template models showing one instance of each of them.

6.1 Hood of car

The first example consists in modelling a surface close to the car's hood of Figure 14.

(a) Front view (b) Right view

Figure 14: The expected surface.

A piece of a possible description of this surface could be:
"*The surface is not very wide. Its main low part, called Z1, is a little arched. The very low part of Z1, called Z2, is very arched ...*".
Figure 15 presents one possible framework answering to the complete description. It shows a structuring tree (a) and one possible location in R_0 of the zones (b) involved in the description where "*Z1 and Z2 are arched, Z3, Z4 and Z5 are bulged and Z6, Z7 and Z8 are three holes*".

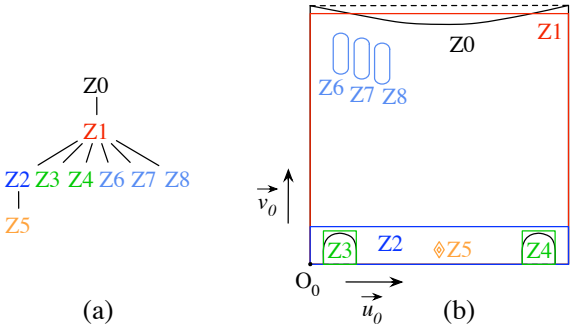


Figure 15: A possible framework.

These data could be stored in a generalised semantic model after the translation process and the completion process. Using this information, the planning process and the construction process give us four template models. These models are very close to each other. The main differences between all of them are around $Z3$ and $Z4$. Figure 16 presents one possible instance of the first resulting template model.

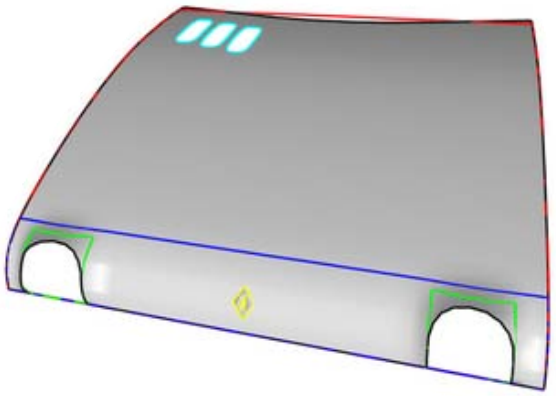


Figure 16: Instance of the first resulting template model with its zones.

6.2 Streamlined Motorcycle

We try to design a part of the streamlined of a motorcycle (see Figure 17.a). This part is illustrated in Figure 17.b.

(a) (b)

Figure 17: (a) Streamlined of the motorcycle and (b) borderlines of the chosen part.

We give our modeller a possible generalised semantic model storing all the requested deformations and zones like it is shown in Figure 18.a. All the resulting solutions are close to the expected surface and can be

considered as fine sketches. Figure 18.b presents one possible instance of the first template model proposed by our system and shows its zones. Figure 18.c illustrates the net of control which is generated to obtain this result.

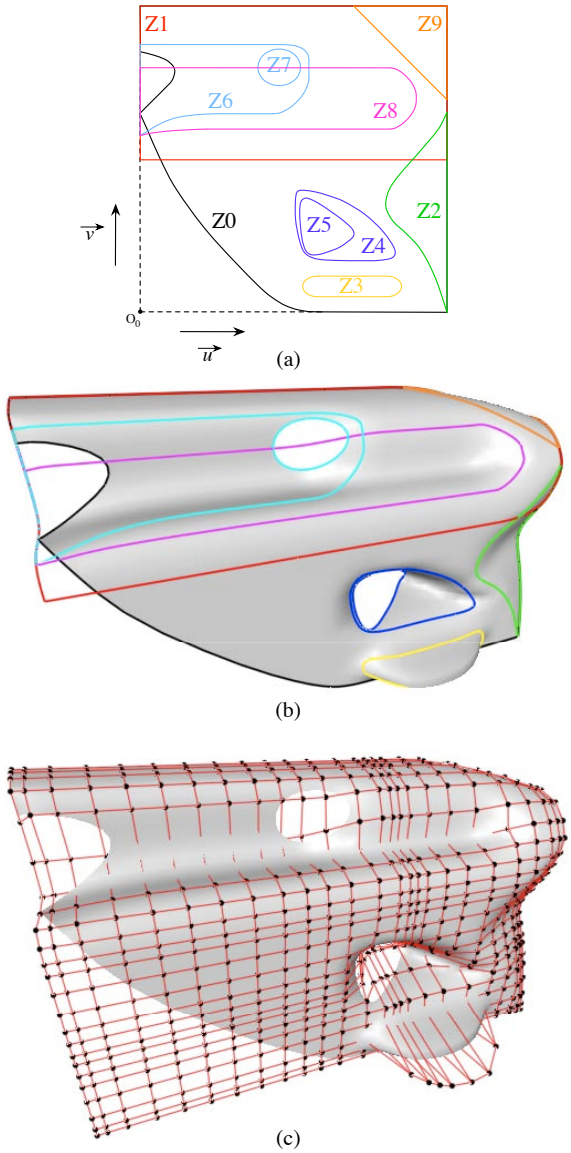


Figure 18: (a) A possible framework corresponding to the expected surface, (b) Instance of the first resulting template model with its zones and (c) its net of control.

7. FUTURE WORKS

We outlined a declarative system able to design parametric surfaces from one or many given descriptions. The implementation of this project is currently in progress and can already give some very interesting results.

The final purpose of our study is to have a complete declarative modeller to design objects using many patches of NURBS surfaces. In order to reach this goal, it is fundamental to develop interfacing modules giving the designer the uttermost means to describe the surface or the object he/she expects. A second very important point is closely related to the declarative approach. Using this method, the user does not need to have any knowledge about the modeller or about the underlying mathematical models. The software has to adapt itself to the designer: a learning process integrated to our system could be a worth feature. A third perspective consists in improving the number of soft and strict constraints currently available in order to be closer and closer to the designer requirements.

We are thinking this tool is a new way of considering modelling process which could be very helpful to designers in a near future.

8. REFERENCES

[BP85] W. Boehm and H. Prautzsch. The insertion algorithm. *Computer Aided Design*, 17(2):58–59, 1985.

[CDMM97] C. Colin, E. Desmontils, J.Y. Martin, and J.P. Mounier. Working modes with a declarative modeler. *Compugraphics*, 1997.

[Dan96] Marc Daniel. Declarative Modeling of fair shapes An additional approach to curves and surfaces computations. In J. Hoschek and P. Kaklis, editors, *Advanced Course on FAIRSHAPE*, pages 77–85. B. G. Teuber Stuttgart, 1996.

[Des00] Emmanuel Desmontils. Expressing constraint satisfaction problems in declarative modeling using natural language and fuzzy sets. *Computers & Graphics*, 24(4):555–568, 2000.

[EH04] Mathieu Estratat and Laurent Henocque. Parsing languages with a configurator. In *European Conference on Artificial Intelligence*, pages 591–595, Valencia, 2004.

[LDB05] R. La Greca, M. Daniel, and A. Bac. Local Deformation of NURBS Curves. In M. Daehlen, K. Morken, and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces Tromso 2004*, pages 243–252. Nashboro Press, Brentwood, TN, 2005.

[Per04] Jean-Philippe Pernot. *Fully Free Form Deformation Freatures for Aesthetic and Engineering Designs*. Phd, Institut National Polytechnique de Grenoble, 2004.

[PT97] L. Piegl and W. Tiller. *The NURBS Book 2nd Edition*. Springer, Berlin, 1997.

Using the Influence of Curve Tangent Vectors to Generate Approximately Uniformly Distributed Reference Points

Mohsen Madi
Department of Computer Science & GIM Research Group
University of Sharjah
Sharjah, UAE, POBox 27272
mmadi@sharjah.ac.ae

ABSTRACT

The need to systematically generate sets of reference points with prescribed arclengths along parametric curves, with accuracy and real-time performance usually arises in applications related to CNC machining, highway and railway design, manufacturing industry, and animation. Mechanisms to produce a parameter set that yield the coordinates of prescribed reference points along the curve $\mathbf{Q}(t) = \{x(t), y(t)\}$ are therefore sought. Arclength parameterizable expressions usually yield the parameter set that is necessary to generate the reference points; however, for typical design curves, such expressions are often not available in closed form. It is desirable to find efficient ways to compensate for lack of arclength parameterization. In this paper, several methods for approximating arclength parameterization are studied. These methods are examined for both accuracy and real-time processing requirements. The paper also introduces a numerical interpolation technique for a cubic interpolator function; the interpolator exploits the influence of end point tangent vectors to generate approximately uniformly distributed reference points as an example application.

Keywords:
Arclength Parameterization, Hermite Interpolation, Reference Points, Bézier Curves.

1 INTRODUCTION

The need to generate sets of reference points ($\mathbf{R}$'s) along paths of mechanical tools or parts is present in CAD and CAM applications. For convenience, reference points are referred to as $\mathbf{R}$'s and the *ith* reference point is designated as $\mathbf{R}_i$. As an example, in CNC machining, computers with CAD systems might be instructed to produce thousands of $\mathbf{R}$'s along the path of a manufactured part (such as the fuselage of an air- plane, where uniform spacing between adjacent reference points is desired to minimize tension) according to specific prescribed locations [Frk92a, Frk96a, Sha82a]. The manipulated part may be required to be affixed to other complementing parts by bolts through adjacent holes, in locations marked by reference points.

Conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

A first step towards generating a set of N $\mathbf{R}$'s with prescribed arclengths is to abstract the physical paths along the object of interest by a *parametric* curve $\mathbf{Q}(t)$ in *Bernstein-Bézier* representation [Far93a]. Next, several problems have to be solved, usually in the following order:

(a) Obtain an expression for the arclength $s(t),\ t \in [0, 1]$.

(b) Compute the total arclength $\mathcal{L} = s(1)$.

(c) Determine the set of desired arclengths $\{s_{i=1,\ldots,N}\}$, $s_i \in (0, \mathcal{L}]$, at which the $\mathbf{R}$'s are to be generated.

(d) Re-parameterize $\mathbf{Q}(t)$ with the parameter set

$\{t_i\}$ obtained from $t(s_i)$: the inverse function of the arclength expression obtained in (a).

Arclength parameterization is desirable because the arclength is an intrinsic quantity of the curve and arclength parameterization is an intrinsic property of the curve. It facilitates design and analysis of curves and surfaces [Bur94a, Gug63a, You93a]. If a closed-form solution is available for items (a) and (d) above, the coordinates of an $\mathbf{R}$ that lie at arclength s_i along $\mathbf{Q}(t)$ may accurately be obtained by first evaluating $t_i = t(s_i)$, and then evaluating $\mathbf{Q}(t)$ at $t = t_i$.

In general, however, because of the non-linearity of the integral expression $s(t)$, it is impossible to solve it in an analytic fashion, and even when this is possible, trying to derive an arclength parameterizable expression $t(s)$ from it usually fails [Sha82a, Frk91a, Frk91b, You93a]. Because of this, several approaches are taken to approximate results that would be attained by arclength parameterization. In this paper, $s(t)$ is exploited to derive a cubic interpolating function to approximate $t(s)$, and the closeness of this function, in comparison to existing methods to actual arclength parameterization, is discussed.

A class of curves known as Pythagorean-hodographs have closed form expressions for their arclengths; however, they still require the solution of a non-linear equation to obtain the parameter as a function of the arclength [Frk91a]. This article is concerned with a more general class of polynomial curves.

The rest of the paper is organized as follows. Sections 2 and 3 present fundamental mathematical preliminaries and related work of several techniques that are geared to addressing approximation of arclength parameterization. Section 4 presents the Cubic Interpolator, an analytical method that embodies Hermite Interpolation to approximate arclength parameterization. This method is also introduced in [Mad04a] and is presented here for convenience. Section 5 shows Experimental Results of all presented techniques presented in a visual comparative fashion for the reader. In Section 6, a new cubic-spline Interpolation Function is derived to produce numerically accurate results when the role of real-time performance is down-played. The paper is concluded in Section 7.

2 MATHEMATICAL PRELIMINARIES

For the purpose of this paper, a curve is represented by a parametric polynomial $\mathbf{Q}(t)$ in Bernstein-Bézier representation:

$$\mathbf{Q}(t_{0\leq t\leq 1}) = \sum_{i=0}^{n} \mathbf{p}_i \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{n-i} t^i. \qquad (1)$$

Properties and importance of such a representation to CAD/CAM are mentioned in the literature [Far93a, Qin89a]. In the above equation, n denotes curve degree, and $\mathbf{p}_i \in I\!E^2$ are the Bézier points that constitute the control polygon of the curve.

The arclength $s(t)$ of $\mathbf{Q}(t)$ is determined by the following integral:

$$s(t) = \int_0^t \|\mathbf{Q}'(\tau)\| \, \mathrm{d}\tau, \qquad (2)$$

where $\mathbf{Q}'(t)$ is the derivative of $\mathbf{Q}(t)$. The total arclength of $\mathbf{Q}(t)$ is therefore $\mathcal{L} = s(1)$.

Because of the non-linearity and the integral term present in $s(t)$, an arclength parameterizable expression $t(s)$ usually has to be approximated, rather then derived directly from (2).

3 RELATED WORK

Several methods have been developed to approximate arclength parameterization, some of which are based on curve dependent tables of data, while others are not. The former class of approximators have the advantage of being adaptable for prescribed accuracy by several numerical techniques. It is difficult to obtain a meaningful comparison for methods which are not of the same class. In some applications such as graphical simulation and animation, where the animated object is to appear at approximately uniformly spaced intervals for smooth appearance, it is the performance rather than the accuracy that is of importance [Mad96a]. In the remainder of this section, an overview of existing methods is presented.

3.1 Basic parametric Flow (BPF)

The simplest method for $\mathbf{R}$'s generation may be called the basic parametric flow (BPF), since a number of N points are produced by uniform parameter spacing (e.g., $t_{i=0,\ldots,N} = i/N$). Although simple and fast, it is well known that this method is not suitable for generating points along the arclength of a curve [Frk97a, Mad96a].

3.2 Sharpe & Thorne (ST)

The method described by Sharpe and Thorne can accurately produce $\mathbf{R}$'s at prescribed arclengths [Sha82a].

However, it has a high computational cost associated with "extracting" the corresponding parametric value for each $\mathbf{R}$ to be generated. Consider the following non-linear equation used to find t_i, the parametric value needed to generate $\mathbf{R}_i$:

$$M(t) = \int_{t_{i-1}}^{t} \sqrt{\mathbf{Q}'(\tau) \cdot \mathbf{Q}'(\tau)}\, d\tau - s_i = 0, \quad (3)$$

where $i = 1..N$, t_{i-1} is the parametric value corresponding to $\mathbf{R}_{i-1}$, the parameter $t = t_i$ is the value corresponding to the next reference-point $\mathbf{R}_i$, and s_i is the arclength from $\mathbf{R}_{i-1}$ to $\mathbf{R}_i$. In order to obtain t_i, a few Newton-Raphson iterations are applied:

$$\tau_j = \tau_{j-1} - \frac{M(\tau_{j-1})}{M'(\tau_{j-1})}, \quad \tau_0 = t_{i-1}, \quad (4)$$

where $j = 1, 2, \ldots, k$, and $M'(\tau_j)$ is the derivative of $M(\tau_j)$. The value of t_i is given by τ_k, where k is the number of iterations required for convergence to an acceptable accuracy.

For applications requiring real-time processing, or those not requiring very accurate spacing of $\mathbf{R}$'s, this method may be impractical.

3.3 Optimal Parameterization (OP)

Farouki's OP is mathematically a rather intricate process [Frk97a]. The given polynomial curve $\mathbf{Q}(t)$ is first transformed into an equivalent rational form by transforming the parameter t in (1) (by applying a Möbius transformation) as follows:

$$t = \frac{(1-\alpha)u}{\alpha(1-u) + (1-\alpha)u}, \quad (5)$$

with $0 < \alpha < 1$ and $0 \leq u \leq 1$. Substituting (5) into (1) results in the following rational form:

$$\tilde{\mathbf{Q}}(u) = \frac{\sum_{i=0}^{n} w_i \mathbf{p}_i \binom{n}{i} (1-u)^{n-i} u^i}{\sum_{i=0}^{n} w_i \binom{n}{i} (1-u)^{n-i} u^i}, \quad (6)$$

where $w_i = (1-\alpha)^i \alpha^{n-i}$. The objective is to find the set of weights $\{w_i\}$ so that u approximates an arclength parameter. The problem is thus to find the "best" α for (6).

While the cost of obtaining the "right" α may be high, this method is better suited for applications requiring real-time processing than ST [Mad96a].

3.4 Cumulative Chordlength (CC)

Cumulative chordlength is a straightforward method to approximate the arclength of a curve. This method can be exploited to generate $\mathbf{R}$'s that visually seem to be uniformly spaced.

The algorithm is as follows: while computing the arclength, the set $\{s_k \,|\, k = 0, 1, \ldots, \eta\}$ (η being the number of chords used to approximate the curve) keeps track of the cumulative chordlength thus far. $\mathbf{R}$'s at distances $\{i\Delta d \,|\, i = 0, \ldots, N;\ \ \Delta d = s_\eta / N\}$, where s_η is the total chordlength, may then be located by searching for their closest values in $\{s_k\}$, and then further refining those values by means of linear interpolation. That is, t_i, the parametric value corresponding to $\mathbf{R}_i$, at distance $i\Delta d$, is approximated by the function $A(i,k)$ as follows:

$$t_i = A(i,k) = \Delta u \left(k - 1 + \frac{i\Delta d - s_{k-1}}{s_k - s_{k-1}} \right), \quad (7)$$

where $s_{k-1} \leq i\Delta d < s_k$, and $\Delta u = 1/\eta$.

While increasing the number of chords approximating $\mathbf{Q}(t)$ *may* increase accuracy, the size of the curve-dependent data-table that has to be maintained also increases [Mad96a].

4 A Cubic Interpolator (CI)

Hermite interpolation [Dav63a, Fol92, Far93a] is used to approximate $t(s)$ for any parametric curve $\mathbf{Q}(t)$. The cubic interpolator (CI) is defined as follows:

$$\mathcal{F}(s) = as^3 + bs^2 + cs + d \approx t(s). \quad (8)$$

An approximation to the inverse function of $s(t) = \int_0^t \|\mathbf{Q}'(\tau)\|\, \mathrm{d}\tau$ may be derived as follows. First, $s(t)$ is differentiated to give

$$s'(t) = \frac{\mathrm{d}s}{\mathrm{d}t} = \|\mathbf{Q}'(t)\|. \quad (9)$$

From (9), $t'(s)$ may be written as follows:

$$t'(s) = \frac{\mathrm{d}t}{\mathrm{d}s} = \frac{1}{\|\mathbf{Q}'(t)\|}. \quad (10)$$

Upon integration of (10), the following results:

$$t(s) = \int_0^s \frac{1}{\|\mathbf{Q}'(\tau)\|}\, \mathrm{d}\tau. \quad (11)$$

The value of $t(s)$ at two parametric values is already known, namely, at $s = 0$ and $s = \mathcal{L}$. Further, for a cubic interpolator, two more items of data are needed to determine values for the four coefficients of $\mathcal{F}(s)$ in

(8): a, b, c and d. The geometry vector at the boundaries of the curve $t(s)$ are obtained as follows:

$$\begin{bmatrix} t(0) \\ t'(0) \\ t(\mathcal{L}) \\ t'(\mathcal{L}) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{\|\mathbf{Q}'(0)\|} \\ 1 \\ \frac{1}{\|\mathbf{Q}'(1)\|} \end{bmatrix}. \qquad (12)$$

Further, by requiring that $\mathcal{F}(s) = t(s)$, and that $\mathcal{F}'(s) = t'(s)$ at the boundaries, we can solve for the coefficients of (8) to get the following solution vector:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \frac{1}{\mathcal{L}^2}\left(c + \frac{1}{\|\mathbf{Q}'(1)\|}\right) - \frac{2}{\mathcal{L}^3} \\ \frac{1}{\mathcal{L}^2} - \frac{c}{\mathcal{L}} - a\mathcal{L} \\ \frac{1}{\|\mathbf{Q}'(0)\|} \\ 0 \end{bmatrix}. \qquad (13)$$

To illustrate the low cost of generating $\mathbf{R}$'s using the CI method, the CI algorithm shown next is an implementation of the method described above ([Mad96a] gives details on implementation and cost of all other algorithms described here). The idea is to generate a set $\{\mathcal{F}_i \mid i = 0, \ldots, N\}$ by the approximating interpolating function, such that evaluation of $\{\mathbf{Q}(\mathcal{F}_i)\}$ renders reference points that are approximately uniformly spaced.

The CI algorithm starts by calculating the coefficients a, b, and c of the cubic interpolating function. The function $\mathcal{F}(s)$ in (8) is evaluated at $\{i\Delta L\}$ to yield $\{\mathcal{L}_i\}$.

```
CI()
    compute L, scale ||Q'(0)|| and ||Q'(1)||
    c ← 1/||Q'(0)||
    a ← c + 1/||Q'(1)|| − 2
    b ← 1 − c − a
    ΔL ← 1/N,  ℓ_0 ← 0
    for i ← 1 to N
        ℓ_i ← ℓ_{i−1} + ΔL
        F_i ← ((aℓ_i + b)ℓ_i + c)ℓ_i
        R_i ← Q(F_i)
END
```

Scaling of $\|\mathbf{Q}'(0)\|$ and $\|\mathbf{Q}'(1)\|$ implies dividing them by $\mathcal{L}$; this scales the whole curve such that $\mathcal{L} = 1$. Note that $\mathcal{F}_i$ above is simply Eq. (8) evaluated using Horner's less costly method.

Note that a precise measure of the deviation from a true uniform distribution is obtainable. Consider, for example, the quintic PH curve of Figure 4, where $N = 80$ reference points are generated along its path. By calculating the distance between every pair of reference points and accumulating the deviation from a true uniform distribution, an exact measure of the deviation from a true uniform distribution is obtained. In this case, the deviation is 5.87%. In the *CI()* function above, such a deviation can be computed by first initializing *deviation* to 0, and by adding the following the following two lines to the for-loop.

$$d_i \leftarrow \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$
$$\textit{deviation} \leftarrow \textit{deviation} + |d_i - \Delta L|$$

Inversely, the distribution is 94.13% accurate, which is a huge achievement over the basic parametric flow (BPF()) parameterization yielding only 69.32%, at more or less the same cost.

5 Experimental Results

The objective of this section is to show how close the $\mathbf{R}$'s generated by the various methods are to the exact $\mathbf{R}$'s (note that uniform spacing is desired), and how the results of the method presented in the previous section compare to those of other methods. In the last subsection, attention is turned to the cost of using the algorithms discussed to generate $\mathbf{R}$'s. Figures 1 to 4 show the result of applying the algorithms discussed to a sample of curves; a much larger sample of curves is found at [Mad96a].

Each of the figures is organized as follows. The actual curve is shown first, followed by plots showing only $\mathbf{R}$'s along their translated paths. In each case, four $\mathbf{R}$ plots are shown: those resulting from BPF (basic parametric flow), ST (Sharpe & Thorne), OP (Farouki's optimal parameterization), and from CI (cubic interpolator).

The ST reference points are considered to be exact (6 to 8 digits of accuracy) and thus are used to compare other results with. Because CC (cumulative chord-length) plots are *visually* indistinguishable from ST plots, they are not shown ([Mad96a] discusses CC and CC plots in more detail).

For symmetric curves (e.g., Figures 1 and 4), the value of α in OP's algorithm is $1/2$, thereby producing results exact to those of BPF [Mad96a, Frk97a]. These and other figures show the closeness of CI results to those of ST, obtained at a considerably lower cost than required by ST.

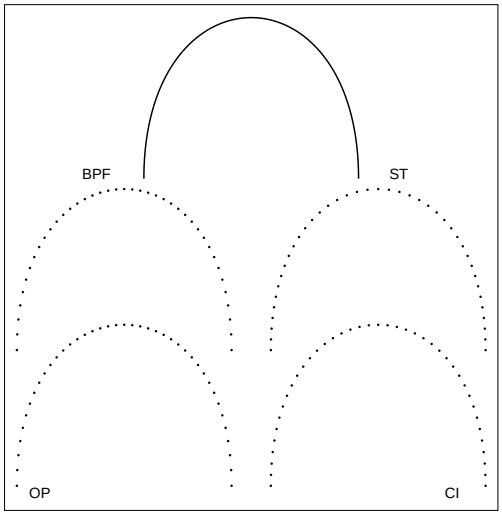


Figure 1: A cubic PH curve.

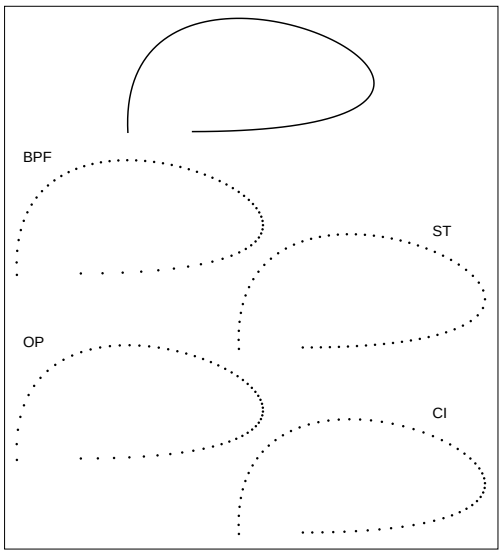


Figure 2: A cubic Bézier with high curvature regions.

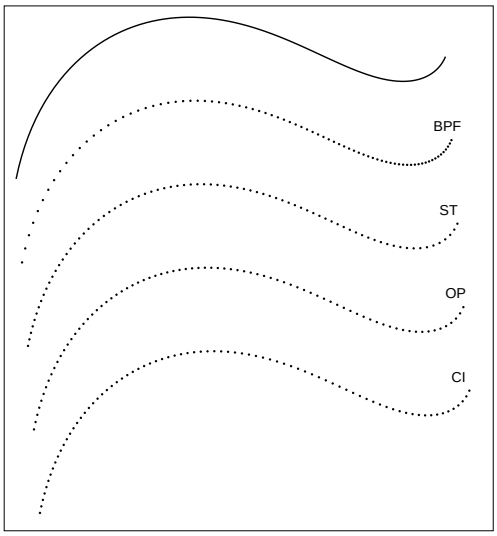


Figure 3: A quintic PH curve.

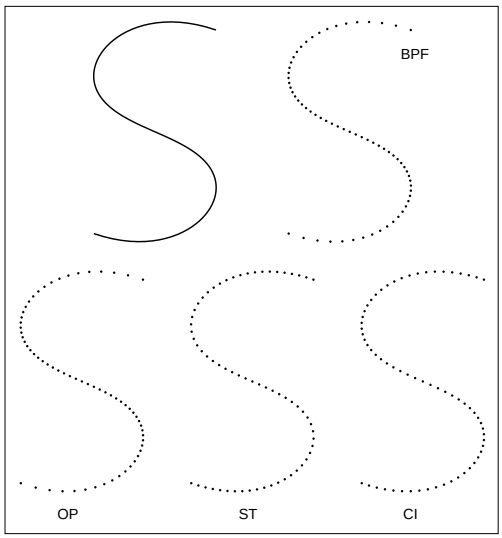


Figure 4: A quintic S-curve.

Refer to [Mad96a, Mad04a] for detailed analysis and tabulated statistics about the run-time cost of each algorithm.

6 USING THE CUBIC-SPLINE INTERPOLATION FUNCTION

In developing the Cubic Interpolator in Section 4, the emphasis was on obtaining an expression for $t(s)$ (that is, t, the parameter of the curve $\mathbf{Q}(t)$, as a function of s, the arclength of the curve) in an analytical fashion, such that, when evaluated at $s_k \in [0, \mathcal{L}]$, the corresponding $t_k \in [0, 1]$ results. It is desirable, for manipulation in mathematical experiments, that it is a single function rather than a piecewise function.

The objective of CI is to make available an approximation to $t(s)$. This is similar to some of the objectives sought in [Sha82a, Gue90a, Frk92a, Frk97a]. However, it is noted that some of these methods depend on analytical expressions, such as the CI or the F-OP algorithms, whereas other methods, such as the ST algorithm, depend on numerical methods to approximate $t(s)$. The first kind of approximators has the advantage of not having to compute and maintain arrays of numbers, or use quadrature techniques to reach a satisfactory result; it is not equitable to compare their accuracy to those that depend on numerical methods. The latter have the advantage that any prescribed accuracy can be obtained by increasing the number of approximating segments to refine results in accordance with some prescribed tolerance, or increasing the number of Newton-Raphson iterations.

In this section, it is shown how the proposed method may be modified so that it also uses numerical techniques to generate reference points (RPs) along parametric curves, thereby making use of the advantages that numerical methods have. The numerical version of the proposed method is called NCI, or, the numerical CI. Its objective is to generate RPs with accuracy and performance comparable to methods which depend on numerical techniques.

6.1 The NCI Derivation & Algorithm

The idea behind developing the NCI is similar to that behind the CC method in Subsection 3.4: the curve $\mathbf{Q}(t)$ is first approximated by $\Im$ segments. The cumulative arclength is calculated at the end of the k*th* segment, $k = 1, 2, \ldots, \Im$, along with the coefficients a_k, b_k, c_k, and d_k. Because each segment is approximated over two Simpson intervals (using Simpson's rule), $\Im$ is always $I/2$, where I is an even number of Simpson intervals used to approximate the $\mathbf{Q}(t)$. This is feasible because Simpson's rule may give the intermediate arclengths at every second function. That is, the arclength of the k*th* segment may be determined by

$$s_k = \frac{1}{3I}(\alpha_{2k-2} + 4\alpha_{2k-1} + \alpha_{2k}) + s_{k-1} \qquad (14)$$

where $k = 1, 2, \ldots, \Im$, $s_0 = 0$, and $\alpha_j = \|\mathbf{Q}'(j/I)\|$. Note that $\mathcal{L} = s_{\Im}$. To determine the coefficients for each segment, Equation (3.27) is rewritten in the following manner:

$$f_k(s) = a_k s^3 + b_k s^2 + c_k s + d_k, \qquad (15)$$

with derivative

$$f'_k(s) = 3a_k s^2 + 2b_k s + c_k, \qquad (16)$$

where $s_{k-1} < s \leq s_k$. For each segment, four equations are required to solve for the four unknowns. Following the same methodology used in Section 3.3.1 (i.e., to formulate four equations by equating each of $f_k(s)$ and $f'_k(s)$ at $s = s_{k-1}$, and $s = s_k$, with the appropriate values of $t(s)$, and $t'(s)$, respectively). Let

$$\begin{aligned} f_k(s_{k-1}) &= \frac{k-1}{\Im}, \\ f_k(s_k) &= \frac{k}{\Im}, \end{aligned}$$

and

$$\begin{aligned} f'_k(s_{k-1}) &= \frac{1}{\alpha_{2k-2}}, \\ f'_k(s_k) &= \frac{1}{\alpha_{2k}}, \end{aligned}$$

the four equations are formulated as follows:

$$\begin{aligned} a_k s_{k-1}^3 + b_k s_{k-1}^2 + c_k s_{k-1} + d_k &= (k-1)/\Im, \\ a_k s_k^3 + b_k s_k^2 + c_k s_k + d_k &= k/\Im, \\ 3a_k s_{k-1}^2 + 2b_k s_{k-1} + c_k &= 1/\alpha_{2k-2}, \\ 3a_k s_k^2 + 2b_k s_k + c_k &= 1/\alpha_{2k}. \end{aligned}$$

The coefficients a_k, b_k, c_k, and d_k may now be solved for, and are as follows:

$$\begin{aligned} a_k &= \frac{1}{(s_k - s_{k-1})^2}\left(\frac{\alpha_{2k-2} + \alpha_{2k}}{\alpha_{2k}\alpha_{2k-2}} - \frac{2}{\Im(s_k - s_{k-1})}\right), \\ b_k &= \frac{\alpha_{2k-2} - \alpha_{2k}}{2\alpha_{2k}\alpha_{2k-2}(s_k - s_{k-1})} - \frac{3}{2}a_k(s_k + s_{k-1}), \\ c_k &= \frac{1}{\alpha_{2k}} - 3a_k s_k^2 - 2b_k s_k, \\ d_k &= \frac{k}{\Im} - (a_k s_k^3 + b_k s_k^2 + c_k s_k). \end{aligned}$$

The complete algorithm follows.

NCI()
  compute $\alpha_{i \leftarrow 0,1,\ldots,I}, \quad s_{k=0,1,\ldots,\Im}$
  scale $\alpha_i,\ s_k$ *so that* $s_{\Im}$ *is unity*
  compute $a_k, b_k, c_k,$ *and* $d_k, k = 0, 1, \ldots, \Im$
  $\Delta L \leftarrow 1/N, \quad k \leftarrow 0, \quad \ell_0 \leftarrow 0$
  for $i \leftarrow 1$ to $N - 1$
    $\ell_i \leftarrow \ell_{i-1} + \Delta L$
    while $\ell_i > s_k$
      $k \leftarrow k + 1$
    $f_i \leftarrow ((a_k \ell_i + b_k)\ell_i + c_k)\ell_i + d_k$
    $\mathbf{r}_i \leftarrow \mathbf{Q}(f_i)$
END

6.2 Visual Results

In this subsection, some RP plots are shown for both the NCI and the CC method, where both will be compared against RPs generated by the ST method. The purpose is to show how the NCI produces better results with fewer segments than is usually required for the CC, and does less computation than is required by the CC, or the ST method. In Figure 5, the circles denote RPs generated by the ST algorithm, while the dots denote RPs generated by the CC and the NCI algorithms. The number next to the algorithm name on each plot indicates the number of chords, segments used by the CC, and the NCI algorithms, respectively.

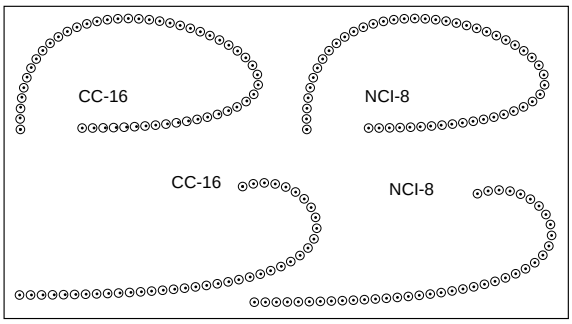


Figure 5: Comparative plots with CC and NCI.

7 Conclusion & Future Research

A survey of methods for approximating the intrinsic arclength parameterization for parametric curves has been presented. The main property of one of the feature methods, the CI, is that it depends on analytical expressions influenced by the tangent vectors at the curve end points, as opposed to methods which depend on numerical techniques. The main advantage of CI is that it is suitable for real-time applications. When the prescribed accuracy to generating the Reference Points is sought over the speed in generating the reference points, a new numerical method, NCI, which depends largely on numerical techniques is derived out of CI and is shown to produce accurate results in a competitive number of iterations.

Work currently in progress iterates over the following points.

- Determine the number of Simpson intervals needed to achieve acceptable accuracy in obtaining the arclength of a curve segment.
- Investigate the performance of an interpolating function of a higher degree. For this, a quintic interpolator will be developed and analyzed against the cubic interpolator to determine whether there is a pay off by using more information at the end points.

References

[Bur94a] Burchard, H.G., J.A. Ayers, W.H. Frey and N.S. Sapidis, Approximating with Aesthetic Constraints, In *Designing Fair Curves and surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*, (N.S. Sapidis, ed.), SIAM, Philadelphia, pp.3–28, 1994.

[Dav63a] Davis, P.J., *Interpolation and Approximation*, Blaisdell Publishing Company, New York, 1963.

[Frk91a] Farouki, R.T. and T. Sakkalis, Pythagorean Hodographs, In *IBM Journal of Research and Development*, Vol. 34, No. 5, pp.736–752, 1991.

[Frk91b] Farouki, R.T. and T. Sakkalis, Real rational curves are not ”unit speed”, In *Computer Aided Geometric Design*, Vol. 8, No. 2, pp.151–157, 1991.

[Frk92a] Farouki, R.T., Pythagorean-hodograph Curves in Practical Use, In *Geometry Processing for Design and Manufacturing*, (R.E. Barnhill, ed.), SIAM, Philadelphia, pp.3–33, 1992.

[Far93a] Farin, G., *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*, Academic Press, Boston, 1993.

[Frk96a] Farouki, R.T. and S. Shah, Real-time CNC interpolators for Pythagorean-hodograph curves, In *Computer Aided Geometric Design*, Vol. 13, No. 7, pp.583–600, 1996.

[Frk97a] Farouki, R.T., Optimal Parameterizations, In *Computer Aided Geometric Design*, Vol. 14, No. 2, pp.153–168, 1997.

[Fol92] Foley, J.D., A. Van Dam, S.K. Feiner and J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison Wesley, Reading, Massachusetts, 1992.

[Gue90a] Guenter, B. and R. Parent, Computing the Arc Length of Parametric Curves, In *IEEE Computer Graphics and Applications*, Vol. 10, No. 3, pp.72–78, 1990.

[Gug63a] Guggenheimer, H.W., *Differential Geometry*, McGraw-Hill, New York, 1963.

[Mad96a] Madi, M.M., *Arclength Approximation for Reference-Point Generation*, Master’s thesis, Dept. of Computer Science, University of Manitoba, June 1996.

[Mad04a] Madi, M.M., Closed-Form Expressions for Approximation of Arclength Parameterization for Bézier Curves, In *Int. J. Appl. Math. Comput. Sci.*, Vol. 14, No. 1, pp.33–41, 2004.

[Sha82a] Sharpe, R.J. and Richard W. Thorne, Numerical Method for Extracting an Arclength Parameterization from Parametric Curves, In *Computer-Aided Design*, Vol. 14, No 2, pp.79–81, 1982.

[Qin89a] Su, B-Q. and D-Y. Liu, *Computational Geometry: Curve and Surface Modeling*, Academic Press, Boston, 1989.

[You93a] Young, E.C., *Vector and Tensor Analysis*, Marcel Dekker, New York, 1993.

Interpolatory Subdivision Curves with Local Shape Control

Carolina Beccari
Dept. of Mathematics
University of Padova
via G.Belzoni 7,
35131 Padova, Italy
beccari@math.unipd.it

Giulio Casciola
Dept. of Mathematics
University of Bologna
P.zza di Porta S. Donato 5,
40126 Bologna, Italy
casciola@dm.unibo.it

Lucia Romani
Dept. of Mathematics
University of Bologna
P.zza di Porta S. Donato 5,
40126 Bologna, Italy
romani@dm.unibo.it

ABSTRACT

In this paper we present a novel subdivision scheme that can produce a nice-looking interpolation of the control points of the initial polyline, giving the possibility of adjusting the local shape of the limit curve by choosing a set of tension parameters associated with the polyline edges. If compared with the other existing methods, the proposed model is the only one that allows to exactly reproduce conic section arcs of arbitrary length, create a variety of shape effects like bumps and flat edges, and mix them in the same curve in an unrestricted way. While this is impossible using existing 4-point interpolatory schemes, it can be easily done here, since the proposed subdivision scheme is non-stationary and non-uniform at the same time.

Keywords
Non-uniform and Non-stationary Subdivision, Curves, Interpolation, Tension, Locality.

1 INTRODUCTION

Curve-subdivision is an iterative process that defines a smooth curve as the limit of a sequence of successive refinements applied to an initial polyline. If, at each refinement level, new points are added into the existing polyline and the original points remain as members of all subsequent sequences, becoming points of the limit curve itself, the scheme is called interpolatory. In 1987 Dyn et al. [Dyn87a] introduced the first interpolatory subdivision scheme for curves, known in the literature as the "classical" 4-point scheme. This name derives from the fact that, starting from a coarse control polygon, the new points recursively introduced between each pair of old points are computed by a linear combination of the immediate four neighboring points. The proposed scheme can generate smooth interpolatory subdivision curves characterized by a tension parameter (denoted by ω) that can only be used to control the shape of the whole limit curve. Another disadvantage of this method is that the global parameter ω acts as a tension parameter only in a very restricted range.

FULL Papers Conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

More precisely, if $0 \leq \omega_1 < \omega_2 \leq \frac{1}{16}$, the limit curve corresponding to ω_2 will be looser than the one corresponding to ω_1. But outside of the range $[0, \frac{1}{16}]$ nothing can be predicted. Recent works in interpolatory subdivision theory [Mar05a, Bec05a, Oht03a] improved the performance of the "classical" 4-point scheme, but, although providing users many choices to fit the different requirements in applications, they still suffer from some limitations to be very useful in future modelling systems. In fact, even though some of them provide degrees of freedom for controlling the shape of the limit curve, no interpolatory schemes have been designed yet to make user manipulations as intuitive as possible and allow the generation of a curve whose behavior can be locally modified in any desirable way. These considerations motivated the research reported in this paper, where we propose a novel scheme that contains [Dub86a, Dyn87a] as special cases and generalizes the non-stationary and uniform interpolatory 4-point scheme in [Bec05a] to possess local shape design parameters which have a visual interpretation on the screen. Due to its versatility, the scheme we are going to present can be used conveniently both for interactive design and for automatic curve fitting. Therefore it is very well-suited for design applications in various fields including Image Processing, CAGD and Computer Graphics.

The paper is organized as follows. In Section 2 we construct and analyze the novel scheme. In Section 3 we investigate its properties and we show its performance. Next in Section 4 we conclude the paper

with some examples and propose some applications for which the novel scheme is very well-suited.

2 DEFINITION OF THE NOVEL INTERPOLATORY 4-POINT SCHEME

In this section we are going to present a novel interpolatory 4-point scheme that allows to adjust the shape of the limit curve by choosing a set of local tension parameters that, when increased within their span of definition, allow to appreciate considerable variations of tension in the corresponding regions of the limit curve. Such tension parameters are associated with the edges of the coarsest polyline and are opportunely updated at each refinement level. In this way, the mask which provides a rule to pass successively from one set of control points to the following, is simultaneously non-stationary (since it is different for each iteration of the subdivision process) and non-uniform (since it also changes along the curve). However, the scheme is very easy to implement because the same rule for generating the new tension parameters is used at each subdivision step.

Consider an oriented polyline $P^0 = \{p_i^0\}_{i\in\mathbb{Z}}$ (where the upper index 0 indicates that no subdivision steps have been applied yet) and assign a tension value $v_{i,0} \in]-1,+\infty[$ to each edge $\overline{p_i^0 p_{i+1}^0}$. The subindex $(i,0)$ underlines that the tension value $v_{i,0}$ is associated with the i-th edge of the polyline defined at the $0-th$ subdivision level. Next, for any $k \geq 1$, update the tension parameters $v_{i,k-1}$ into $v_{i,k}$ through the relation

$$v_{i,k} = \sqrt{\frac{1+v_{i,k-1}}{2}} \tag{1}$$

and successively compute the coefficients

$$w_{i,k} = \frac{1}{8v_{i,k}(1+v_{i,k})} \tag{2}$$

that will be used to map the polygon $P^{k-1} = \{p_i^{k-1}\}_{i\in\mathbb{Z}}$ to the refined polygon $P^k = \{p_i^k\}_{i\in\mathbb{Z}}$ by applying the following subdivision rules:

$$\begin{aligned} p_{2i}^k &= p_i^{k-1} \\ p_{2i+1}^k &= -w_{i,k}\,(p_{i-1}^{k-1} + p_{i+2}^{k-1}) \\ &+ \left(\frac{1}{2} + w_{i,k}\right)(p_i^{k-1} + p_{i+1}^{k-1}). \end{aligned} \tag{3}$$

Remark 1. Since after each round of subdivision one new point is inserted between two old ones, every edge of the old control polygon P^{k-1} is split into two new edges in the refined one. Thus, denoted by $v_{i,k-1}$ the tension parameter associated with the edge $\overline{p_i^{k-1} p_{i+1}^{k-1}}$, since such an edge is split into the two new edges $\overline{p_{2i}^k p_{2i+1}^k}$, $\overline{p_{2i+1}^k p_{2i+2}^k}$, according to (1) we will make them inherit respectively the tension values $v_{i,k}$ that is $v_{2i,k} = v_{2i+1,k} = v_{i,k}$.

Remark 2. Note that the initial tension values $v_{i,0}$ are updated before computing the coefficients $w_{i,1}$ used in the first subdivision step. Hence, for any choice of $v_{i,0} \in]-1,+\infty[$, (1) implies that $v_{i,k} \in]0,+\infty[\ \forall k \geq 1$ and (2) leads to $w_{i,k} > 0\ \forall k \geq 1$.

2.1 Convergence analysis

In this paragraph we are going to show that our novel interpolatory subdivision scheme always generates a C^0-continuous limit curve. More precisely, we will prove that, given an initial polyline P^0, the subdivision rules in (3) define an increasingly dense collection of polylines P^k that converge to a continuous curve. To show this, we will exploit two important properties of our scheme described in the following lemmas.

Lemma 1. *Given the initial parameter $v_{i,0} \in]-1,+\infty[$, the recurrence relation in (1) satisfies the property:*

$$\lim_{k\to+\infty} v_{i,k} = 1. \tag{4}$$

Proof 1. To prove this we recall that a monotonic and bounded sequence is always convergent and in particular: if it is non decreasing and upper bounded, then it converges to the upper bound of the values it assumes, whereas if it is non increasing and lower bounded, then it converges to the lower bound of the values it assumes. For the sequence defined by

$$\begin{cases} v_{i,0} \in]-1,+\infty[\\ v_{i,k} = \sqrt{\frac{1+v_{i,k-1}}{2}} \quad \forall k \geqslant 1 \end{cases} \tag{5}$$

it holds:

- if $v_{i,0} = 1$, then the sequence $\{v_{i,k}\}_{k\geq 1}$ is stationary;
- if $v_{i,0} \in]-1,1[$, then the sequence $\{v_{i,k}\}_{k\geq 1}$ is non decreasing;
- if $v_{i,0} \in]1,+\infty[$, then the sequence $\{v_{i,k}\}_{k\geq 1}$ is non increasing.

Therefore, in all these cases $v_{i,k}$ is convergent and converges to 1. In fact, called ℓ its limit, we have

$$\ell = \lim_{k\to+\infty} v_{i,k} = \lim_{k\to+\infty} \left(\sqrt{\frac{1+v_{i,k-1}}{2}}\right) = \sqrt{\frac{1+\ell}{2}}.$$

Thus, solving the last equation with respect to ℓ we get $\ell = 1$. □

Lemma 2. *Due to the recurrence (1), the parameters $v_{i,k-1}$ and $v_{i,k}$ satisfy the relation*

$$\frac{1-v_{i,k}}{1-v_{i,k-1}} < \frac{1}{2} \tag{6}$$

for any $v_{i,k} \in]0,+\infty[$, $k \geqslant 1$.

Proof 2. Exploiting recurrence (1) we can write the ratio $\frac{1-v_{i,k}}{1-v_{i,k-1}}$ in the following way

$$\frac{1-v_{i,k}}{1-v_{i,k-1}} = \frac{1-\sqrt{\frac{1+v_{i,k-1}}{2}}}{1-v_{i,k-1}} = \frac{1}{2+\sqrt{2}\sqrt{1+v_{i,k-1}}}$$

and observe that since $\sqrt{2}\sqrt{1+v_{i,k-1}} > 0 \; \forall k \geqslant 1$, thus

$$\frac{1-v_{i,k}}{1-v_{i,k-1}} < \frac{1}{2}. \qquad \square$$

Now, exploiting the well-known theorem in [Dyn95a], that relates the convergence of a non-stationary scheme to its asymptotically equivalent stationary scheme, we are going to show the following result.

Proposition 3. *The locally controlled scheme in (3) converges and generates C^0-continuous limit curves.*

Proof 3. To prove our thesis we need to show that, given an initial polyline, the novel locally controlled scheme defined by the following mask

$$m^{k-1} = \left[-\frac{1}{8v_{i-1,k}(1+v_{i-1,k})},\ 0,\ \frac{(2v_{i,k}+1)^2}{8v_{i,k}(1+v_{i,k})},\right.$$
$$\left. 1,\ \frac{(2v_{i+1,k}+1)^2}{8v_{i+1,k}(1+v_{i+1,k})},\ 0,\ -\frac{1}{8v_{i+2,k}(1+v_{i+2,k})} \right] \quad (7)$$

converges to a C^0-continuous limit curve.
Since from (4) it follows that

$$m^{\infty} \equiv \lim_{k\to+\infty} m^{k-1} = \left[-\frac{1}{16},\ 0,\ \frac{9}{16},\ 1,\ \frac{9}{16},\ 0,\ -\frac{1}{16}\right],$$

then the non-stationary subdivision scheme associated with (7) converges to the stationary scheme in [Dub86a], that is exactly the "classical" 4-point scheme in [Dyn87a] obtained with $\omega = \frac{1}{16}$.
Next, by computing $m^{k-1} - m^{\infty}$ we have that

$$\begin{aligned} \| m^{k-1} - m^{\infty} \|_{\infty} &= \frac{|1-v_{i-1,k}|(2+v_{i-1,k})}{16v_{i-1,k}(1+v_{i-1,k})} \quad (8)\\ &+ \frac{|1-v_{i,k}|(2+v_{i,k})}{16v_{i,k}(1+v_{i,k})} \\ &+ \frac{|1-v_{i+1,k}|(2+v_{i+1,k})}{16v_{i+1,k}(1+v_{i+1,k})} \\ &+ \frac{|1-v_{i+2,k}|(2+v_{i+2,k})}{16v_{i+2,k}(1+v_{i+2,k})} \end{aligned}$$

where $v_{i-1,k}, v_{i,k}, v_{i+1,k}, v_{i+2,k} \in]0,+\infty[\ , \forall k \geqslant 1$.
Now, by the well-known theorem in [Dyn95a], if

$$\sum_{k=1}^{+\infty} \| m^{k-1} - m^{\infty} \|_{\infty} < \infty, \quad (9)$$

then the two schemes are asymptotically equivalent, and since m^{∞} is C^0, we can conclude that the scheme associated with m^{k-1} is C^0 too. Thus, to show our thesis we only have to prove that relation (9) holds as the parameters $v_{i-1,k}$, $v_{i,k}$, $v_{i+1,k}$, $v_{i+2,k}$ vary in the interval $]0,+\infty[$. For space limitations, we just give a brief sketch of the proof here. Observe that, from the analytic properties of the series, once we have proved the convergence of each term in (8), relation (9) will immediately follow. To this aim, thanks to the results in lemma 1 and 2, we can repeat, for each term of the sum in (8), the calculations presented in [Bec05a] and claim that the schemes m^{k-1} and m^{∞} are asymptotically equivalent, which proves our thesis. $\square$

Remark 3. Note that the proposed scheme is asymptotically equivalent to the "classical" interpolatory 4-point scheme in [Dyn87a] with parameter $\omega = \frac{1}{16}$, that is C^1. Hence, since according to [Dyn95a], the continuity level of a non-stationary scheme is, in general, the same as that of the stationary scheme to which this scheme converges, we expect the limit curve generated by the scheme in (3) to be C^1.

3 THE LOCAL TENSION PARAMETERS

The novel scheme described by (3) has a very powerful and intuitive geometric interpretation. If we write the second line of (3) in the form

$$\begin{aligned} p^k_{2i+1} &= \frac{p^{k-1}_i + p^{k-1}_{i+1}}{2} \quad (10)\\ &+ 2w_{i,k}\left(\frac{p^{k-1}_i + p^{k-1}_{i+1}}{2} - \frac{p^{k-1}_{i-1} + p^{k-1}_{i+2}}{2}\right) \end{aligned}$$

it is clear that the new point p^k_{2i+1} turns out to be the mid-point of the edge $\overline{p^{k-1}_i p^{k-1}_{i+1}}$, corrected by a vector $2w_{i,k}e$, where e denotes the vector connecting the mid-points of the edges $\overline{p^{k-1}_i p^{k-1}_{i+1}}$ and $\overline{p^{k-1}_{i-1} p^{k-1}_{i+2}}$ (see Figure 1).

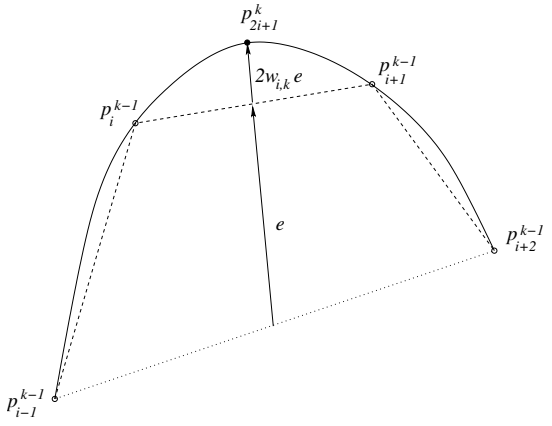


Figure 1: Geometric interpretation of the local parameter $w_{i,k}$.

The value of $w_{i,k}$ in (10) clearly depends on the choice of the initial tension parameter $v_{i,0}$. For different values of $v_{i,0}$ ranging from -1 to $+\infty$, we will have different values of $w_{i,1}$ in $]0,+\infty[$. The important cases to be mentioned are $v_{i,0} = 1$ ($w_{i,1} = \frac{1}{16}$) and $v_{i,0} \to +\infty$ ($w_{i,1} \to 0$). In the first case, in fact, the subdivision scheme becomes stationary and the limit curve we obtain coincides with the one generated by [Dub86a] and by [Dyn87a] choosing the parameter ω equal to $\frac{1}{16}$. In the second case, instead, the portion of curve confined to the endpoints of the i-th edge is pulled towards the linear interpolant of those two points, since when $w_{i,1}$ is exactly zero, the scheme exactly generates the linear interpolant to the initial control points.
In particular, progressively increasing the value of $v_{i,0}$, the portion of curve confined to the endpoints of the i-th edge will become progressively tighter and tighter.
It is interesting to note also that, by setting all initial tension parameters equal to the same value v_0, we get the uniform tension-controlled interpolatory scheme defined in [Bec05a].

3.1 Local tension parameters provided by the user

The local tension parameters $v_{i,0}$ that define the interpolatory scheme presented in Section 2, can be either arbitrarily provided by the user to intuitively model the limit shape, or automatically set to let the limit curve fit special requirements in applications.
In Figure 2 we show an example of interactive design where the local tension parameters play an important role in the overall display of the final shape. The top left curve has been obtained by setting all the parameters $\{v_{i,0}\}_{i=0,\ldots,6}$ to $\gamma = \cos(\frac{2\pi}{7})$. As explained in [Bec05a], such a configuration of tension values, applied to the regular heptagon denoted by the dashed line, generates the exact circle interpolating its vertices $\left(\cos(\frac{2j\pi}{7}), \sin(\frac{2j\pi}{7})\right)$, $j = 0,\ldots,6$. The consecutive five interpolants demonstrate the role of the local tension parameters specified in the array v, by showing their influence on the resulting shapes. Comparing the six figures, we can see that, by opportunely modifying the tension values γ, we can obtain different ways of achieving local shape control on the limit curve.
These results and many others, obtained using different initial polylines and various local tension parameters, confirm our conjecture about the C^1-smoothness of the curve generated by the refinement scheme in (3).
Additionally, they point out that the model we have proposed exhibits many properties that a subdivision scheme should include to become useful for geometric modelling applications. Indeed, it allows:

- **Intuitive shape deformations**
  Among all the shape parameters that we can find in existing subdivision models, the ones that provide a tension effect indeed appear totally intuitive and possess a direct visual interpretation on the screen (see Figure 2).

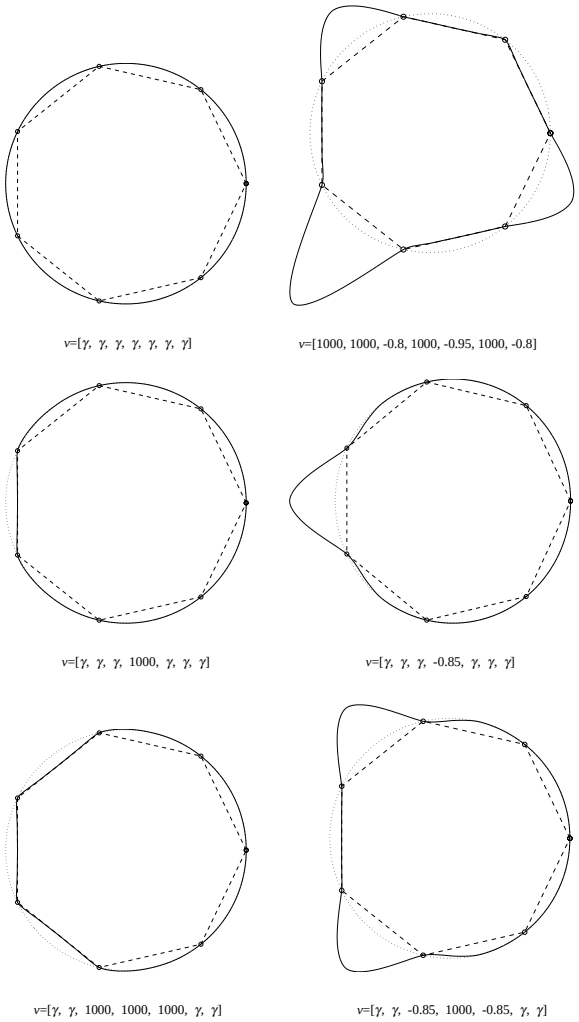


Figure 2: Closed subdivision curves produced by interpolation of the vertices of a regular heptagon using the tension parameters in the indicated array v where $\gamma = \cos(\frac{2\pi}{7})$.

- **Local control**
  Each tension parameter, associated with a specified edge of the initial polyline, influences the shape of the limit curve only in a restricted zone, corresponding to the related edge and the neighboring parts of its two adjacent edges. This allows a very powerful local shape control (see Figure 2).

- **Warping, flattening and mixed effects**
  Warping is a tool which can be used to deform a local segment of a curve pulling out a bump; flattening is a tool which allows to easily introduce straight line segments into curves. The local tension parameters can reproduce both warping and

flattening behaviors as well as provide a variety of interesting shape effects. The subdivision curves that we can generate, in fact, are able to incorporate both round shapes and flat shapes, like bumps and localized flat edges. Additionally such effects can be mixed in the same curve in an unrestricted way, always allowing soft transitions between them (see Figure 2).

3.2 Local tension parameters automatically set

In this subsection we will show the capability of the proposed subdivision scheme to fit certain classes of curves widely used in CAGD applications, like polynomials, conic sections and piecewise cubic Bézier curves.

Since an interpolatory subdivision process defines a smooth curve as the limit of a sequence of successive refinements applied to an initial polyline, the approximating algorithm we are going to describe requires to select an adequate set of points on the reference curve C to start the locally controlled refinement procedure described in Section 2. Note that, the greater the number of selected points is, the more the approximating shape tends towards the original curve C, but the more the shape descriptor size and the number of computations increase. To balance these two extreme scenarios, the strategy adopted for determining the initial polyline to which apply the refinement process should select the minimum number of points needed to get a visually precise reconstruction. By the term *visually precise* we mean that, if you compare our interpolatory subdivision curve with the original one, you can hardly spot the differences between them (see Figures 4, 5, 6, 7).

Let $P^0 = \{p_i^0\}_{i\in\mathbb{Z}}$ be the initial sequence of points opportunely sampled on C. Due to (10), the points p_{2i+1}^1 computed at the first subdivision step, will be placed on the line r passing through the mid-points of the edges $\overline{p_i^0 p_{i+1}^0}$ and $\overline{p_{i-1}^0 p_{i+2}^0}$, at a distance from $\frac{p_i^0+p_{i+1}^0}{2}$ inversely proportional to the tension value $v_{i,0}$. Therefore, in order to achieve a visually precise fitting, each point p_{2i+1}^1, defined in correspondence of the i-th edge $\overline{p_i^0 p_{i+1}^0}$ of the coarsest polygon P^0, should be placed exactly on the curve C. According to this request, the initial tension value $v_{i,0}$ will be determined in such a way the point p_{2i+1}^1 turns out to be the intersection point between the line r and the curve C. Repeating this procedure for each curve segment, we will be able to automatically determine a tension value $v_{i,0}$ that, used in the refinement process (3), generates the interpolatory subdivision curve that approximates the given curve C. Although we have no guarantee that in the following refinement levels the subdivision process will insert all the new points on the curve C, whenever we start from a sequence of points $P^0 = \{p_i^0\}_{i\in\mathbb{Z}}$ suitably sampled on C, all our experiments showed visually precise reconstructions.

The following algorithm implements an automatical procedure to get an interpolatory subdivision curve that turns out to be visually precise in respect to the original curve C.

Algorithm 1

1. *Determine the initial polyline* $P^0 = \{p_i^0\}_{i\in\mathbb{Z}}$*;*
2. *compute the tension parameters* $v_{i,0}$ *associated with the edges* $\overline{p_i^0 p_{i+1}^0}$*, in such a way the points* p_{2i+1}^1*, inserted through (3), lie on* C *(for each edge of the initial polyline, this requires to solve the system of equations deriving by forcing the point* p_{2i+1}^1 *to be placed on* C*);*
3. *for* $k = 1$ *to the desired refinement level*
   - 3.1 *compute the tension values* $v_{i,k}$ *through (1);*
   - 3.2 *compute the coefficients* $w_{i,k}$ *through (2);*
   - 3.3 *apply the refinement rules (3);*
4. *stop.*

Remark 4. Note that, due to the rule in the second line of equation (3), to insert a new point p_{2i+1}^k in the refined polyline P^k, it is necessary to possess a well defined two-neighborhood (given by two points on its left and right side in the coarsest polyline P^{k-1}). Hence, if C is an open curve, the initial polyline P^0 should be extended to contain two more sample points both at the beginning and at the end of the initial sequence $P^0 = \{p_i^0\}_{i\in\mathbb{Z}}$.

In the following we will show how the proposed algorithm leads to some useful applications of our locally controlled interpolatory 4-point scheme.

The first one is related to two desirable properties of subdivision schemes, namely polynomial precision and conics reproduction. These two terms usually refer to the capability of exactly reproducing a specified curve by applying the subdivision scheme to a set of control points uniformly sampled on it. As far as we know, existing interpolatory subdivision schemes are able to reproduce neither polynomials nor conic sections starting from unevenly sampled points.

Exploiting Algorithm 1 we can show that our interpolatory scheme turns out to be exact for linear polynomials even if the initial data are unevenly sampled on them. Additionally, a visually precise fit to points unevenly sampled on any quadratic or cubic polynomial and any arbitrary conic section, can be achieved too (see Figure 3).

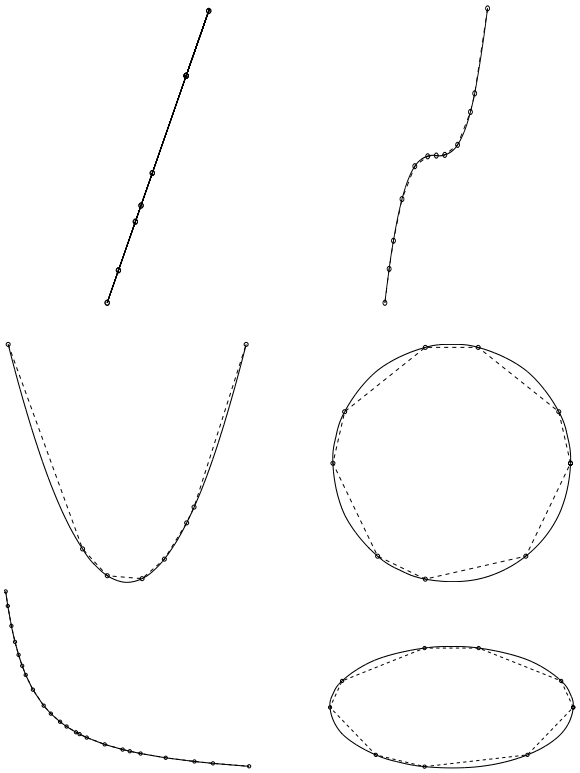


Figure 3: Representation of polynomials and conic sections via the interpolatory subdivision curve with local shape control (dashed lines: polylines; solid lines: subdivision curves).

On the other hand, it can be easily verified that, when we consider sample points uniformly spaced, the proposed scheme is exact for cubic polynomials. In fact, performing the computations described in Algorithm 1, we get that the same tension parameter $v_{i,0} = 1$ should be applied on the whole curve. This value coincides with the global tension parameter introduced in [Bec05a] to exactly reproduce cubic polynomials. Analogous results can be found repeating the same computations when the initial data are evenly sampled on a conic section [Bec05a]. Therefore, when starting from uniformly distributed points, we can conclude that by applying Algorithm 1 we can work out the initial tension parameters that allow the subdivision scheme in (3) to be exact for cubic polynomials and to exactly reproduce all conic sections.

Since in CAGD applications curves are frequently represented in the piecewise cubic Bézier form, we want to show now an additional application of our algorithm which allows to obtain a very good representation of a curve C in the piecewise cubic Bézier form. Denoted by Q^0 the sequence of control points defining the piecewise cubic Bézier curve C, we select from Q^0 the subsequence consisting exactly of all the junction points of the single Bézier segments. Next, to define the initial polyline P^0 for generating the interpolatory subdivision curve, we add supplementary points wherever two consecutive selected points (so far computed) turn out to be too distant. According to our experiments this strategy always ensures a visually precise reconstruction.

4 APPLICATIONS

The proposed subdivision scheme was designed to make our model a candidate of choice for many applications. An example consists in proposing this novel class of interpolatory subdivision curves as an efficient and economical representation of the outline curves used to describe fonts as well as output images of tracing algorithms [Sel03a]. In general, a vector outline describes a digitalized image via a family of piecewise cubic Bézier curves that may join either C^0 or C^1 continuously. Our subdivision scheme provides a mathematical description of vector outlines that turns out to be simpler and more efficient than the one currently used in Postscript files and tracing algorithms. In fact, although in vectorial outlines one switches frequently between round shapes and flat shapes, by using only one subdivision curve with local shape control, we will be able to make a good job (see Figure 4).

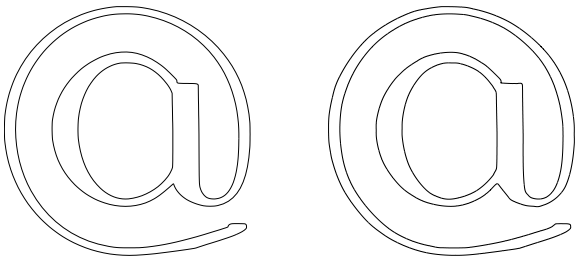


Figure 4: Two piecewise cubic Bézier curves describing the outline of the character "@" - 73 cubic segments (left); two single-piece interpolatory subdivision curves - 75 points, 73 tension parameters (right).

Remark 5. Note that, due to the smoothness properties of the scheme, to make the job perfectly, and then allowing the possibility of reproducing sharp corners between flat regions and round ones, we need to generate distinct subdivision curves and stitch them together in the corners (see Figure 5).

Now we point to a number of advantages that could make this novel class of interpolatory subdivision curves the standard representation of vectorial outlines.

The primary advantage of interpolatory subdivision curves with local shape control over piecewise cubic Bézier curves has to do with the physical storage

Figure 5: Two piecewise cubic Bézier curves describing the outline of the character "d" - 24 cubic segments (left); the corresponding piecewise interpolatory subdivision curves - 9 single-piece subdivision curves, 33 points, 24 tension parameters (right).

of vector outlines in files. In fact, as the set of local tension parameters act as handles to model the contour that best fits the original image, exploiting a small number of points per contour, we can obtain a very compact numerical format for representing outline curves. Since the necessary information to represent the outline curve are 1 endpoint and 1 tension parameter per segment, we manage to remarkably compress the data to be stored in the file and optimize them for speed and for handling contour images with large character sets.

Another consistent advantage is that our technique turns out to be also very convenient for computing the points required for the rasterization of the final curve, which consists in converting the outline to a pattern of dots (whether it's screen pixels or the dots of a laser, inkjet or wire-pin printer) on the grid of the output device.

All these observations let us notice that indeed there are several key improvements which may swing the future in curve-subdivision's favor. While subdivision surfaces have not been widely adopted by the CAGD community [Gon01], maybe some day subdivision curves could intelligently substitute the use of piecewise cubic Bézier curves in describing Postscript fonts as well as the output images of tracing algorithms.

Exploiting the procedure described in Algorithm 1, very good results have been obtained over a range of outline images. We show here the reconstructions we got by testing the algorithm on the data obtained from the outline of a Type1, cmr10 Postscript font (Figures 4, 5), an hand-drawn sketch (Figure 6) and a detail of a university logo (Figure 7). As it appears, if you compare Figures 4, 5, 6, 7 (right) with the corresponding outlines of piecewise cubic Bézier curves in Figures 4, 5, 6, 7 (left), you can hardly spot the differences between them.

5 CONCLUSIONS AND FUTURE WORK

Stimulated by the observation that a "good" local interpolatory scheme does not seem to be presently available, we have proposed a novel subdivision scheme for interpolatory curves which, in contrast to existing models, gives the possibility of generating a big variety of good quality curves modifying the tension parameters associated with the edges of the initial polyline. To our knowledge, this is the first interpolatory subdivision scheme, easy to implement and computationally economical, that includes many classical properties as well as several original features considered vital or simply desirable in applications. For example, the proposed scheme can generate a limit curve that can be locally modified with the help of shape control parameters. Namely, it is able to produce a locally controlled interpolatory curve which can incorporate a variety of shape effects like bumps and flat edges, and can mix them in an unrestricted way, always allowing a soft transition between them. Additionally, instead of being altered by the user for interactive design purposes, the tension parameters can be automatically set in such a way the limit curve generated can elegantly fit curves extremely used in geometric modelling systems, such as conic section arcs and piecewise cubic Bézier curves. As an application, we have proposed our interpolatory subdivision curves with local shape control as an efficient and economical representation of the vector outline of a scanned image or a letter-form shape. This makes it possible to introduce subdivision techniques in document description languages as Postscript and encourages the CAGD community to swing in curve-subdivision's favor.

The authors are looking, as a future work, to extend the curve subdivision scheme to surfaces. This could be quite useful for various applications in different fields of studies.

6 ACKNOWLEDGEMENTS

This research was supported by MIUR-PRIN 2004 and by University of Bologna "Funds for selected research topics".

7 REFERENCES

[Bec05a] Beccari C., Casciola G., Romani L., A non-stationary uniform tension controlled interpolating 4-point scheme reproducing conics. Submitted to Comput. Aided Geom. Design, July 2005, available at www.dm.unibo.it/∼casciola/html/publications.html.

[Dub86a] Dubuc S., Interpolation through an iterative scheme. J. Math. Anal. Appl. 114 (1986), 185-204.

Figure 6: Outline of Loxie and Zoot cartoon: 236 piecewise cubic Bézier curves - 3767 cubic segments (left); reconstruction via interpolatory subdivision curves with local control - 236 single-piece subdivision curves, 4003 points, 3767 tension parameters (right).

Figure 7: Outline of a detail of the University of Bologna logo: 561 piecewise cubic Bézier curves - 6679 cubic segments (left); reconstruction via interpolatory subdivision curves with local control - 561 single-piece subdivision curves, 7240 points, 6679 tension parameters (right).

[Dyn87a] Dyn N., Levin D., Gregory J.A., A 4-point interpolatory scheme for curve design. Comput. Aided Geom. Design 4 (1987), 257-268.

[Dyn95a] Dyn N., Levin D., Analysis of asymptotically equivalent binary subdivision schemes. Math. Anal. Appl. 193 (1995), 594-621.

[Gon01] Gonsor D., Neamtu N., Can subdivision be useful for geometric modeling applications? Boeing Technical Report ♯01-011.

[Mar05a] Marinov M., Dyn N., Levin D., Geometrically controlled 4-point interpolatory schemes. Advances in Multiresolution for Geometric Modelling, Dodgson N.A., Floater M.S. and Sabin M.A. (eds), Springer-Verlag 2005, 301-315.

[Oht03a] Ohtake Y., Belyaev A., Seidel H.-P., Interpolatory subdivision curves via diffusion of normals. Proceedings of the Computer Graphics International (CGI'03) IEEE Computer Society (2003), 1-6.

[Sel03a] Selinger P., Potrace: a polygon-based tracing algorithm, September 2003, unpublished, available at http://potrace.sourceforge.net/potrace.pdf.

Real-time rendering of complex surfaces defined by atlas of discoids

Benoît Piranda Sylvain Magdelaine Didier Arquès
SISAR Team
University of Marne La Vallée
5, boulevard Descartes, Champs sur Marne
77454, Marne La Vallée, France
piranda@univ-mlv.fr ewik@free.fr arques@univ-mlv.fr

ABSTRACT

This paper expounds a new method of complex surfaces rendering permitting to visualize atlas of discoids in real-time. The atlas of discoids allow to define surfaces very easily without the topological constraints we could get with the classical methods such as polygons meshes ones. Our basic model is completed by many algorithms permitting either to reconstruct implicit surfaces or to calculate the global illumination using radiosity algorithm. But up to now, no fast viewing method has been proposed for this modeling algorithm.
We present here a real time rendering algorithm which exploits the recent extensions of OpenGL library and the possibilities offered by GPU like the shaders programs to perform non standard fragments mixing operations.

Keywords

Surface reconstruction, real-time visualization, atlas of discoids.

1. INTRODUCTION

Computer graphics techniques propose a large choice of solutions for surface modeling. The most widely used method consists in defining any surface by a polygonal mesh [Fol90]. Even if a classical structure like the winged-edge data structure [Sil94] allows to represent efficiently such a mesh, many drawbacks exist. In a rendering process, visual artifacts due to preponderant directions (the edges of the polygons) appear. Manipulation such as patch subdivisions become complex (see for instance [Man88] for a complete overview about solid modeling) because topological constraints between neighboring patches have to be maintained. We can also point out that similar topological problems appear in the surfaces reconstruction process of from a set of points [Boi84].

An interesting alternative consists in using Spline [Bar89] or Nurbs [Roc89] surfaces which model any surface by a collection of piecewise-polynomial patches instead of previous planar ones. More complex surfaces can be modeled but the main drawbacks concern continuity constraints in the junction points [Wat93] and rendering computation time generated by ray-tracing algorithm.

Different interesting works aim at combining the previous approaches. Szeliski and Tonnesen [Sze92] propose to use disks or particles to represent the mesh describing an implicit surface. Oriented particles interact each other according to repulsion and attraction forces to automatically treat modifications of the surface (split, join, extend). Indeed, Witkin and Heckbert [Wit94] use oriented particles to sample regularly an implicit surface.

The recent point-base rendering model allows to visualize a surface described by a set of points. Splats defined in [Zwi01] are commonly represented by disks centered on these points, and that may overlap each others. These splats are projected onto the screen and filtered by a Gaussian kernel in order to represent a continuous textured and illuminated surface.

Other recent works [Arq00a] present another method using elements of surfaces (discoids) that can overlap each others. So, defining such a surface with a set of disk-like patches imposes few constraints. For example, it avoids completely topological constraints

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

associated to a classical mesh. On the contrary, overlapping areas are welcomed. We just have to place surface elements in the area we want the surface to exist, and we choose the geometry and the size of each discoid in order to obtain a "total covering" of the surface (cf. figure 1).

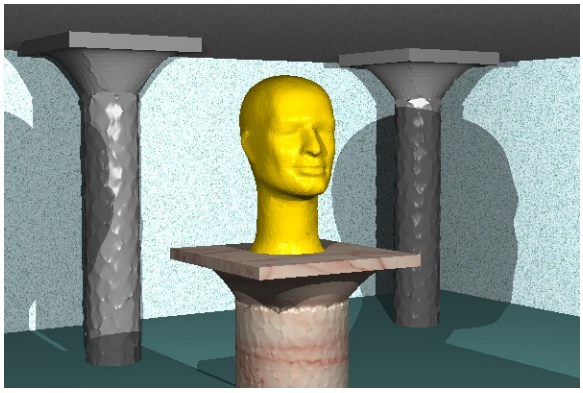


Figure 1: scene completely defined by atlas of discoids.

In [Arq00b], this geometrical model is completed by a rendering algorithm based on the radiosity global illumination model. However, in the two former papers, the visualization of surfaces described by atlas of discoids suffers the drawbacks of the ray-tracing algorithm, a calculation time that doesn't allow real-time utilization.

The following image shows a human face made of only 19 more general discoids defined by overlapping polynomial surfaces.

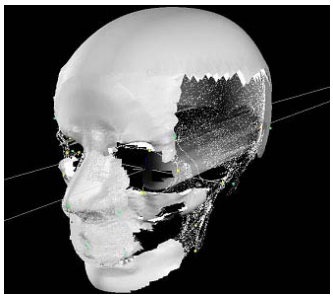


Figure 2 : surface defined by 19 discoids.

Recent new capacities of the OpenGL library in programming the Z-Buffer algorithm allow to intervene in the heart of the rendering process developing short programs : the shaders. One of the categories of these short programs, called "fragment shaders", allows to modify the algorithm that selects the visible polygon in order to determine the color of the filled pixel.

In this paper, we propose a new method permitting to visualize surfaces reconstructed from an atlas of simple surfaces that overlap each other in real-time using recent capacities of the OpenGL library in programming the Z-Buffer algorithm. We first present the atlas of discoids model, then we show how use shaders programs to calculate the melting of overlapping discoids. Then, we present some results on images.

2. ATLAS OF DISCOIDS AND Z-BUFFER

The atlas of discoid

On a given surface *S*, the atlas system allows to reconstruct any interest function *F*, for instance a temperature or luminance function from an atlas of discoids. We define an atlas of discoids as a set of *N* disks-like patches $\{D_i, i=1\ldots N\}$ that cover entirely but approximately the surface *S* without taking into account overlapping problems. (cf. figure 3).

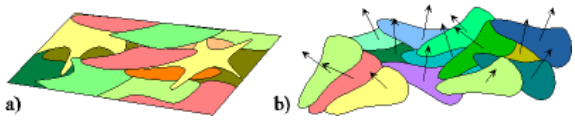


Figure 3 : example of planar and complex surfaces defined by an atlas of discoids.

Only for understanding reasons and notation simplifications, we just consider in this section the case of a planar surface *S*. Any point *M* of *S* is covered by a subset of discoids as shown in figure 4. Then the value of the interest function *F* in *M* is defined by :

$$F(M) = \sum_{i/M\in D_i} \alpha_i(M) F_i(M) \qquad (1)$$

where :

$F_i(M)$ is the local interest function defined for each point of the disk D_i covering *M*;

α_i is a merging operator, defined and positive on each disk D_i, and which verifies $\sum_{i/M\in D_i} \alpha_i(M) = 1$.

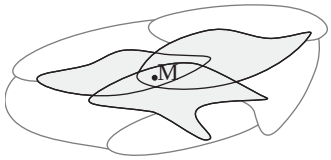


Figure 4 : only gray disks are used to define the interest function in M.

The choice of the function α_i for each disk D_i is relatively free. It can be seen as a decomposition of the interest function in a function base with a local geometrical support: the discoid. An interesting method consists in choosing any set of positive functions β_i and to define α_i by:

$$\alpha_i(M) = \frac{\beta_i(M)}{\sum_{j/M\in D_j} \beta_j(M)} \qquad (2)$$

For example, the β_i functions may depend on the distance from *M* to the center of the discoid. By

choosing a function β_i which varies continuously from 1 in the center to 0 in the border of each disk, it is easy to verify that the merging operator runs as a smoothing operator only in the overlapping area.

In this paper, we propose to define the β_i function over the discoid with a gray level texture for each discoid (cf. figure 5). Black points are null values of β_i and the white ones correspond to $\beta_i = 1$.

Just notice that null values of the β_i define the boundaries of the discoid.

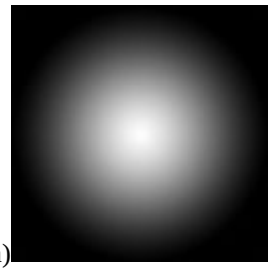
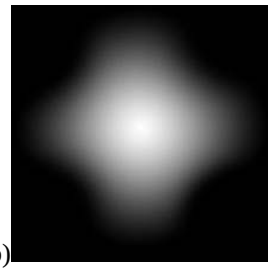
a)  b) 

Figure 5: two shapes of discoids defined by a texture : a disk (a) and a star (b)

Z-Buffer and fragment

First, let's remind the well-known Z-buffer algorithm, and its implementation in the OpenGL library. The Z-Buffer algorithm consists in painting every polygon of the scene. For each pixel filled by the polygon, the process emits a "fragment" that contains the geometrical information and illumination coefficients of the polygon. One of these fragments is selected to be used to color the corresponding pixel in the color buffer.

Fragment shaders programs allow to develop our own treatments applied to each fragment. For example, we have developed a short fragment program that calculates the illumination applying Phong [Pho75] illumination model in the level of the fragment. By using this fragment program instead of classical OpenGL lighting program, we obtain a smoother surface aspect as shown on figure 6.

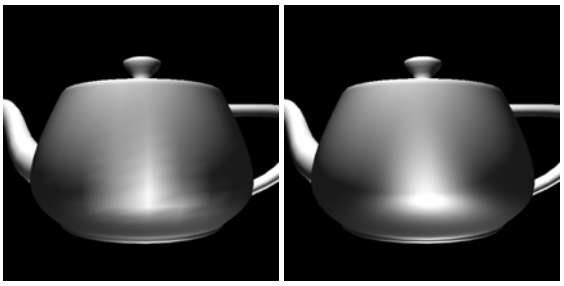


Figure 6: the same model rendered using OpenGL method (left) and with fragment program (right).

3. ADAPTATIONS FOR REAL-TIME VISUALISATION

Geometrical considerations

If we consider now the general case, the disks do not exactly cover the surface *S*. We have to associate to the point *M* and for each disk D_i a point M_i. Equation (1) becomes:

$$F(M) = \sum_{i/M \in D_i} \alpha_i(M_i) F_i(M_i) \qquad (3)$$

Using the Z-Buffer algorithm to render the surface, we mix the points M_i projected on the point M of the surface. If we consider a point of view *O* and the direction of viewing $\vec{u}$, the points M_i correspond to the intersection point between the ray $[O\vec{u})$ and the discoid D_i as shown in the figure 7.

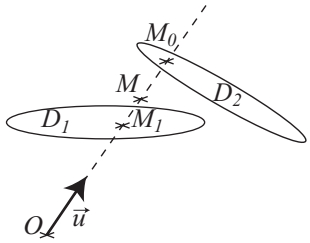


Figure 7: association of points M_i of discoids to point *M* of the surface.

Then during the Z-Buffer process, the pixel on with the point M is projected receives a fragment for each discoid that covers *M*. Then, knowing the value β_i (memorized in the gray level texture) for the *n* discoids M_i projected in M, we deduce from equation (3):

$$F(M) = \frac{\sum_{i=1}^{n} \beta_i(M_i) F(M_i)}{\sum_{i=1}^{n} \beta_i(M_i)} \qquad (4)$$

In the Z-Buffer algorithm, the denominator part of the equation (5) is only known at the end of the process, ie when all fragments are arrived. So, in order to calculate the interest function of the surface as soon as the fragments arrive, we express the equation (4) under a new incremental formulation:

$$\begin{cases} F^{(0)}(M) = 0 \\ F^{(t)}(M) = \dfrac{F^{(t-1)}(M) \sum_{i=1}^{t-1} \beta_i(M_i) + \beta_t(M_t) F^{(t)}(M)}{\sum_{i=1}^{t} \beta_i(M_i)} \end{cases} \qquad (5)$$

Where *t* is the order of the fragments' arrival.

Level of details

More precisely, a problem of level of details appears. Considering all discoids that produce a fragment for the same pixel, we have to melt only discoids that effectively cover the same point of the surface. We don't have to mix far away discoids as shown in figure 8 and presented in a practical case in the image of figure 12 (where the two skulls are faraway).

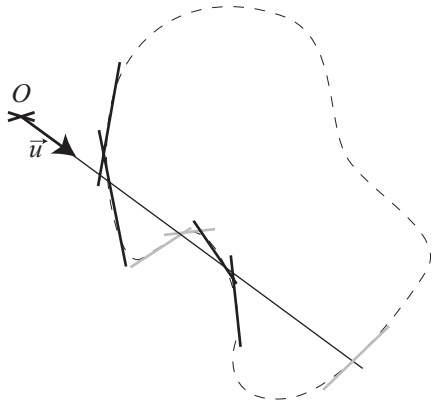


Figure 8: criteria of selection of overlapping discoids

Two criteria are evaluated: the orientation of discoids and the distance between discoids along the vision axis. We use OpenGL culling test to delete fragments corresponding to the bad-oriented discoids (in gray in figure 8) and we use an adapted depth test to select which fragments should be mixed. We melt only the discoids that are very near to the memorize surface: when the depth *Z* of a new fragment is very different to the memorized depth Z_{mem}, we apply the classical algorithm of Z-Buffer.

More precisely, we first calculate the depth map of the scene without melting discoids and save it in a texture (called "depth" in the following code). Then during the other steps of the program, we only treat the discoids placed near this visible surface. The threshold is a level of details parameter, it must be chosen much smaller than the dimension of the discoids.

```
// texture memorizing the depth buffer
uniform sampler2D depth;
// texture of beta values
uniform sampler2D discoid;
// threshold value
uniform float t;

void main (void)
{ float beta = texture2D(discoid,
                    vec2(gl_TexCoord[0])).a;
  float dpt=texture2D(depth,
                 vec2(gl_FragCoord/size)).x;
// depth buffer
  if(beta==0.0 || gl_FragCoord.z > dpt+t)
  { discard;
  }
}
```

For example, in figure 9 the same scene is drawn with a excessive (left) and good (right) value of threshold. We can observe some artefacts due to the mixing of too far away discoids.

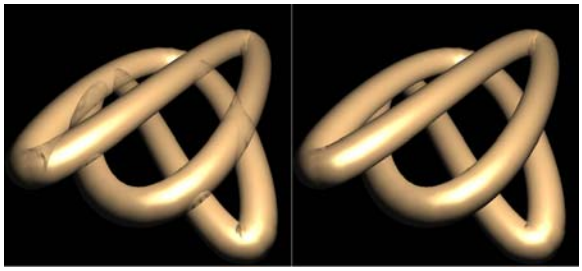


Figure 9: a scene rendered with two different values of threshold

Illumination model

The illumination model used in our implementation is based on Phong illumination model [Pho75] applied on each fragment. The luminance L is calculated using the following expression:

$$L=K_a+K_d\cos(\alpha)+K_s\cos^N(\gamma) \qquad (6)$$

Where K_a is the ambient term, K_d the Lambertian diffusion coefficient, K_s the specular reflexion coefficient, and N the shininess Phong coefficient.

The first developed solution doesn't use the MRT (Multi Render Target) library, so we have to write illumination parameters in only 4 bytes (RGBA) per fragment. So we use two times the equation (5) with $\cos(\alpha)$ and $\cos^N(\gamma)$ as interest function. After incremental calculation, the result is placed in the RVBA memory according to the following table. The two last bytes are reserved to memorize the denominator part of equation (5).

R	G	B	A
$\sum_{i=1}^{t}\beta_i(M_i)\cos\alpha_i$	$\sum_{i=1}^{t}\beta_i(M_i)\cos^N\gamma_i$	$\sum_{i=1}^{t}\beta_i(M_i)$	

Table 1. illumination data of a fragment

It is also possible to directly calculate the $\sum_{i=1}^{t}\beta_i(M_i)$ term using the blending capacities of the OpenGL library. Each fragment sends its β_i value coded in two bytes of the fragment color (first 4 bits in A and last 4 bits in B) and then the blending process computes the sum in the color buffer.

```
// texture containing beta values
uniform sampler2D discoid;
void main (void)
{ // beta value on the fragment
  float beta = texture2D(discoid,
vec2(gl_TexCoord[0])).a;
  float coef = beta*256.;
  gl_FragColor.b = floor(coef/16.);
  coef-=gl_FragColor.b*16.;
  gl_FragColor.b/=256.;
  gl_FragColor.a = coef/256.;
}
```

We just have to copy this buffer in a texture in order to use it in the next steps of the program. The value of the $\sum_{i=1}^{t} \beta_i(M_i)$ term is finally obtained combining the B and A components of the texture : Bx16 + A.

```
// texture containing encoded sumBeta values
uniform sampler2D sumBeta;

void main (void)
{ // sBcode.ba : encoded values of sumBeta for the
current fragment
  vec4 sBcode = texture2D(sumBeta, vec2(gl_FragCoord
/ size));
  float sumBeta = sBcode.b*16. + sBcode.a;
}
```

The following figure shows a sphere defined by 20 discoids, with the beta function presented on figure 5a. The first image (on left) shows the placement of the patches carrying the discoids. The image on the right is obtained by melting the previous discoids using our algorithm and applying a per fragment Phong lighting program.

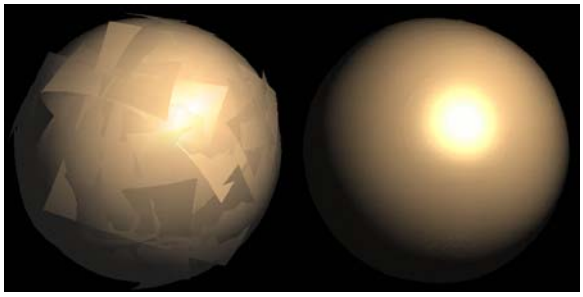


Figure 10: A sphere covered by 20 curved discoids

4. RESULTS

The two following examples presented below show surfaces described by atlas of discoids rendered by our program. The frame rate of these two examples is about 20 images per second on a NVidia Geforce 6 6800 GT Graphic card.

Figure 11: Eagle model defined by 8800 discoids

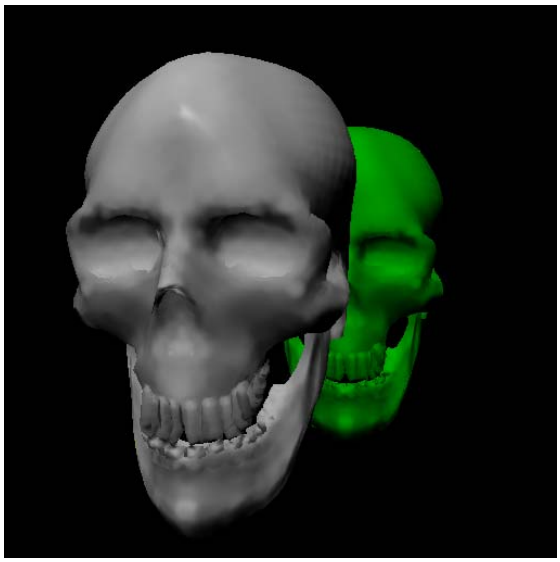


Figure 12: Two skulls by 7176 planed discoids each.

5. CONCLUSION AND FUTURE WORKS

In this paper we propose a new algorithm permitting to visualize in real-time, surfaces defined by an atlas of discoids. This approach reduces topological constraints by simply describing any object by an atlas of discoids.

Such method combined to a real-time visualization process is the first step to the development of a surface modeler that creates continuous surface from natural drawing of shapes.

With the last new capacities proposed by OpenGL 2.0, and more precisely MRT (Multiple Render Target), it is possible to memorize many interest functions for each discoid, written in different buffers. Moreover, the using of floating point textures allows better precision of calculation. These improvements will permit to construct surfaces associated to more illumination parameters and better speed of calculation.

6. REFERENCES

[Arq00b] D. Arques, S. Michelin and B. Piranda, Overlapping radiosity: using a new function base with local disk support, WSCG'2000, vol. 3, 2000, pp. 236-243.

[Arq00a] D. Arquès, S. Michelin, B. Piranda. Modelisation of Implicit Surfaces Driven by an Atlas of Discoids, GraphiCon'2000, 2000.

[Bar89] R.H. Bartels and C.J. Beatty, A technique for the direct manipulation of spline curves, Graphics Interface'89, 1989, pp. 33-39.

[Boi84] J.D. Boissonnat, Geometric structure for three dimensional shape representation, ACM

Transactions on Graphics, vol. 3(4), 1984, pp. 266-286.
[Fol90] J. Foley, A. Van Dam, S. Feiner and J. Hugues, Computer graphics: principles and practice, 2nd edition, Addison Wesley, 1990.
[Man88] M. Mäntylä, An introduction to solid modeling, Computer Science Press, 1988.
[Pho75] B. T. Phong. Illumination for computer generated pictures. Communication of the ACM, Vol.18, n°6, 311-317, 1975.
[Roc89] A. Rockwood, K. Heaton and T. Davis, Real-Time Rendering of Trimmed Surfaces, SIGGRAPH '89 Proceedings, vol. 23(3), 1989.
[Sil94] F.X. Sillion and C. Puech, Radiosity and global illumination, Morgan Kaufmann publisher, 1994.
[Sze92] R. Szelisky and D. Tonnesen, Surface Modelling with Oriented Particle Systems, Computer Graphics, vol. 26(2), 1992, pp.185-194.
[Wat93] A. Watt, 3D Computer Graphics, 2nd edition, Addison Wesley, 1993
[Wit94] A.P. Witkin and P.S. Heckbert, Using Particles to Sample and Control Implicit Surfaces, SIGGRAPH '94 Proceedings, 1994, pp. 269–277.
[Zwi01] M. Zwicker, H. Pfister, J. van Baar and M. Gross, Surface splatting, In proceedings of ACM SIGGRAPH 2001 (2001), pp. 371-378.

Introducing artistic tools in an interactive paint system

Koen Beets Tom Van Laerhoven Frank Van Reeth

Hasselt University - Expertise Centre for Digital Media
and transnationale Universiteit Limburg
School of Information Technology
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{koen.beets, tom.vanlaerhoven, frank.vanreeth}@uhasselt.be

ABSTRACT

While paint systems have been around for a long time, systems capable of capturing the complex behavior of paint media like watercolor, gouache, Oriental ink, oil and acrylic paint have emerged only recently. However, concentrating on the simulation of paint and brush mechanics, these applications mostly provide just a minimal set of instruments assisting users in creating artwork.

We report on the extension of our physically-based paint system for watery paint with a set of versatile tools supplying users with more control during the painting process. We introduce, among others, the use of masking fluid, a special-purpose brush using patterns to steer paint diffusion, and the adoption of an absorbent, textured piece of paper to remove some wet paint from the canvas. Results show that images created with genuine paint, using real-life counterparts of some of these tools, can be closely reproduced with our application. Additionally, our digital tools can produce effects that are difficult or impossible to achieve with real paint, while retaining the spontaneous nature of the resulting images.

keywords: I.3.4 [Graphics Utilities]: Paint systems, I.3.5 [Computational Geometry and Object Modeling]: Physically based modeling, B.6.1 [Design Styles]: Cellular arrays and automata

1 INTRODUCTION

The large amount of research invested in creating digital equivalents of real-life painting is not surprising, given the fact that throughout the majority of our history people have been fascinated with creating images. The activity is accessible and very simple so that users of all ages can practice it; yet, mastering it is complex enough that it remains challenging, even to an expert. Digital interactive paint systems have the same extensive audience.

Working with an ideal paint system would have several advantages compared to the real process:

- the material is free, durable, and can easily be customized according to a user's needs;
- painting becomes even more accessible;
- making images is more practical; there is very little set-up time, no mess and no cleaning up afterwards;
- it allows post-processing and making perfect copies;
- during painting, digital tools allow all sorts of non-physical operations, undoing, removing pigment and water, ...

Conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

Despite this convincing list of advantages, so far paint systems have had little impact on how artists make images. Most still prefer to use the traditional methods because the digital results look too artificial. Also, while moving from a real-life canvas to the computer screen, an artist should gain an incredible amount of flexibility, which does not apply to a great deal of the applications the relatively long history of painterly rendering research has brought us.

In recent years, however, researchers are targeting exactly these problems, which already resulted in interactive systems that are able to capture the complex behavior of Oriental ink, acrylic and watercolor paint. Most of these systems provide just a minimal set of instruments complementing the brush and canvas model. Also, while trying to mimic the real-life equivalents of the virtual material, the possibilities of integrating tools that are non-physical or quasi-physical are not fully explored yet. In this work we propose several of these extensions to our simulation for watery paint; some have no correlation with the real world, while others are commonly used by artists but had not been introduced in any paint system yet.

The rest of this work is organized as follows: first we will state our contributions to the domain, followed by a short discussion on previous related work. Next, a quick review of our paint system for thin paint is presented. Section 5 discusses how some of the techniques we will add to the system are applied in real-life, with right after some notes on how we integrated them. Finally, results are shown and discussed in section 7 along with some conclusions and future work in section 8.

2 CONTRIBUTIONS

We extend our real-time interactive paint system for watery paint with the following tools:

Textured tissue removes pigment and water with a textured, absorbent piece of paper, leaving a distinct pattern.

Diffusion controller allows customizing or "steering" the diffusion process by means of a user-defined pattern.

Masking fluid sits on the canvas, protecting paint in all layers beneath it.

Selective eraser removes some/all pigment and/or water from an active paint layer.

Paint profile collects predefined palette and canvas parameters under a single identifier, facilitating the ability to switch between different paint media and styles.

With these increments we are able to create some distinctive effects that previously were difficult or impossible to produce.

3 BACKGROUND

In literature, several authors recognized the fact that interactive painting applications are in need of supporting tools, complementing the procedure of putting paint or ink on a canvas using some sort of brush model. This issue arrises within the domain of interactive painting systems, rather than the "automatic" painterly rendering systems that convert an input image to a painterly style equivalent.

Smith stressed the fact that one should make a difference between a digital paint *program* and a digital paint *system* [Smi01]. Although both can be described as just "applications processing pixels", the term "system" implies many more features and much more capability than just a "program" that simulates painting with a brush on a canvas. A paint system might even encompass several other sub-programs.

Related to this matter, Baxter *et al.* pointed out that current research in the area of painterly rendering mostly emphasizes the *art* of painting, the appearance of the final product, while paying less attention to the *craft* of painting [BSLM01], the way an artist uses different materials to express himself. His dAb paint system for painting with acrylic or oil combines a deformable 3D brush model with a force feedback input device to enhance the user's sense of realism, who can focus on the painting process itself. Using a minimalistic interface, however, it disregards some of the extra digital benefits that can assist a user during the creation of his artwork.

On the other side of the support-spectrum are the commercial paint systems like Adobe Photoshop and Corel Painter. The amount of complex tools integrated with these applications is overwhelming, making it virtually impossible for untrained artists to create digital paintings with the same ease as in the real world.

The recent work of Chu *et al.* on Oriental ink simulations [CT05] incorporates several special effects and controls that are hard or impossible to reproduce in real life. They provide control over paint wetness, allowing dried ink to flow again, or to fast-forward the drying process. An interesting option in their system is the possibility to tinker with the pigment advection algorithm, creating magnetic-like effects where pigment can be pushed or pulled. A "splash and spray" tool provides an alternative way of applying ink to the canvas, by emitting a pattern of ink drops using some simple physics scheme.

4 A CANVAS MODEL FOR SIMULATING WATERY PAINT

In this section we will give a brief overview of our recently introduced canvas model for interactive simulation of thin, watery paint in real-time [VLVR05].

Inspired by an implementation of a system that produced watercolor images from input images [CAS+97], it is the first model that provides real-time painting experience with watercolor paint, targeting both realistic paint behavior and convincing rendering of semi-transparent glazes. In addition, it supports simulation of paint media related to watercolor, like gouache and Oriental ink.

Figure 1 depicts a schematic view on the canvas model consisting of three active layers and an unlimited number of passive layers. The passive layers are considered to be layers of dried paint that do not participate anymore in the dynamics simulation.

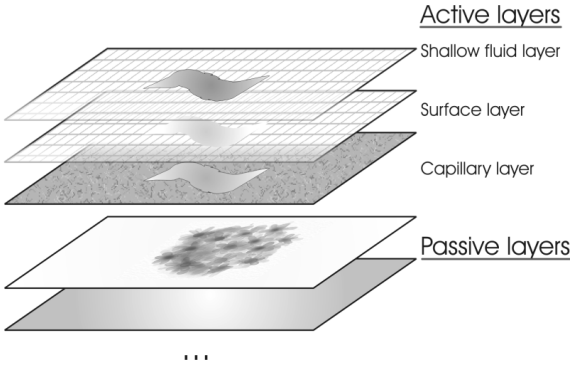


Figure 1: The canvas model, with three active layers and an unlimited number of passive layers.

Each layer consists of a 2D grid of cells on which operations take the form of cellular automata. In the

shallow fluid layer, the model adopts stable fluid dynamics algorithms based on the work of Stam [Sta99] to transfer pigment densities and water quantities on top of the canvas. Heuristic rules handle the deposition of pigment within the irregularities of the canvas surface (represented by the surface layer), as well as the evaporation, absorption and capillary diffusion of water inside the canvas structure (in the capillary layer).

Rendering of the canvas is handled by the Kubelka-Munk diffuse reflectance model.

All algorithms participating in the dynamics phase and the rendering phase of the simulation are implemented on graphics hardware as fragment shaders that operate on a set of textures containing the relevant simulation data.

For more information concerning the details of this model, we refer to literature [VLVR05].

5 REAL-WORLD ARTISTIC TOOLS

In this paragraph we describe how the techniques we implemented in our system correspond to working with water colour in real-life.

5.1 Textured tissue

In real-life, an absorbing piece of paper, a sponge, or even dried breadcrumbs can be used to remove an excess of pigment and water from the canvas and to leave an impression of the texture. This can be done right after applying the paint, but also when the paint is already dry. In that case, the dried paint is made wet again by rubbing over it using a wet brush. This way, some paint pigments come loose again, and can adhere to the absorbing piece of paper. The result is a print of the texture of the paper, which can create interesting visual effects. Figure 3 shows the application of an absorbing piece of paper.

5.2 Diffusion controller

The second contribution of this paper is a diffusion controller, which allows steering the diffusion of paint, using a special-purpose brush. In this way, we can obtain blooming colours also obtainable in real-world painting when painting using a technique known as “wet-in-wet”. The English artist Joseph Mallord William Turner was the first to exploit extensively the wet-in-wet technique, where paint is applied on a wet piece of paper [Mac05]. Combined with the transparency effects that can be obtained with glazing, blooming can create vibrating visual effects.

In real-life, the amount of blooming depends on the wetness of the paper, on the pigment characteristics and on paint quality. The directions and amount of blooming can be influenced by holding the paper in the desired direction, because it is influenced by gravity, and by the amount of paint and water. Blooming is often used for skies and water in particular, but also to obtain interesting effects of atmosphere, light, and texture.

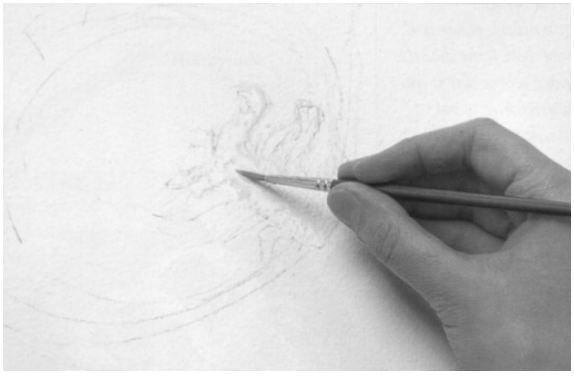


Figure 2: Applying masking fluid with a brush on the canvas [Smi98].

5.3 Masking fluid

Masking can be useful to prevent staining a specific area of the canvas, in several ways. A first application is to obtain sharp and exact contours in the painting. A second application is to protect large areas from paint when using broad washes.

Note that masking can be used both in areas with and without paint, and that it can be used in layers. In real-life painting, masking can generally be done using two different materials: tearable adhesive tape, and masking fluids.

A first method is to delineate a certain area of the paper by using adhesive tape, which can be teared to obtain specific shapes. This method however does not allow for very precise control. A second, and more precise, method is applying masking fluid with a brush on the canvas (figure 2). This fluid is a removable colorless liquid made from rubber and latex staining that can be used to mask areas of work that need protection when color is applied in broad washes. Once dry, these areas remain protected and cannot be penetrated by colour. Afterwards, the dry masking fluid can be removed by rubbing with the hand.

In our system, the masking fluid is more easily and precisely controllable. The tool retains its usefulness, even with the availability of Undo functionality, and the use of several layers. With the masking fluid, one can easily obtain sharp borders throughout several layers of paint, whereas it would be cumbersome to get this effect without.

5.4 Selective eraser

This tool has no exact counterpart in real world painting. There, it is only possible to remove part of the pigments which are deposited on the paper. Precise control is not possible, and selectively removing a specific pigment is not possible either.

5.5 Paint profile

In real life, a popular technique is "mixed media", which means a painting is created using a combination of different media, e.g. both Chinese ink and water colour, and even gouache. Using these profiles, it becomes much easier to integrate different media into a single painting.

6 DIGITAL ARTISTIC TOOLS

We will now take a closer look on how the techniques introduced in the previous section are implemented.

6.1 Textured tissue

The tissue is applied to areas on the canvas where paint is not completely dry yet. At this point the brush becomes a metaphor for the finger, applying pressure on the tissue as long as the brush touches the canvas surface.

The tissue is modeled as a texture in which one color channel is used for storage of a height field (the irregular surface of the tissue), and a second color channel marks transferred cells on the tissue, making sure the tissue does not remove all material from the canvas.

Water in the fluid layer of the canvas model is absorbed by the tissue and, at the same time, an amount of pigment is transferred from the canvas surface area to the tissue according to the magnitude of the pressure at that point. In our examples we used a circular pressure field with a Gaussian distribution. The amounts of material removed are not individually stored by the tissue but combined to a single "mark"-value. This procedure is shown in figure 5.

The operation is implemented using three fragment shaders. Two shaders remove pigment and water amounts from the canvas according to the tissue texture's x-channel, containing height field values between 0 and 1 (the irregular surface). A third shader marks the transferred cells in the y-channel of the tissue texture.

We assume that a "clean" tissue is used after lifting the brush from the surface. At this time, the marked cells on the tissue are cleared.

The left side of figure 4 depicts a close-up of the tissue used in our system. It was sampled from the piece of paper displayed in the center part of figure 3. The final result of this operation is a footprint of the tissue, as shown in the right part of figure 4.

6.2 Diffusion controller

The "diffusion controller" introduces per-cell blocking values in the simulation, forcing pigment densities to follow a predefined pattern during the diffusion process. There is no real-world tool that allows this kind of paint manipulation.

The simulation step that deals with the diffusion of pigment densities is handled by the diffusion component $\nu\nabla^2\vec{p}$ of the Navier-Stokes equation [Sta03]. Because of its simplicity and cheap execution on graphics hardware, we use the Jacobi iteration method to solve this Poisson equation and calculate the new pigment densities p^{new} at cell (i, j) each time step [VLVR05]:

$$p_{i,j}^{\text{new}} = (\alpha p_{i,j} + p_{i+1,j} + p_{i,j+1} + p_{i-1,j} + p_{i,j-1})/(4.0+\alpha) \quad (1)$$

with $\alpha = \text{cellarea}/(\nu\Delta t)$ and ν the diffusion rate. Intuitively, what happens is that all cells keep exchanging content, trying to cancel out differences in concentration with neighbouring cells (figure 7). This way, when dropping pigment densities into a wet area on the canvas, this particular diffusion algorithm would result in something similar to the top image in figure 6, which is not what we want and which certainly does not correspond to what happens in real-life (section 5.2).

If we introduce a scale factor $\beta_{i,j}$ at each point where cells exchange content, we can control the amount of pigment density that is passed from one cell to a neighbour while retaining conservation of mass:

$$p_{i,j}^{\text{new}} = (\alpha p_{i,j} + (\beta p)_{i+1,j} + (\beta p)_{i,j+1} + (\beta p)_{i-1,j} + (\beta p)_{i,j-1})/(\beta_{\text{sum}} + \alpha) \quad (2)$$

with $\beta_{\text{sum}} = \beta_{i+1,j} + \beta_{i,j+1} + \beta_{i-1,j} + \beta_{i,j-1}$.

A blocking factor $\beta_{i,j} = 0.0$ effectively cancels out pigment exchange with neighbouring cells, while $\beta_{i,j} = 1.0$ corresponds to no blocking at all. These values are stored in a texture (the blocking texture) and used by the fragment shader that implements the diffusion algorithm to look up per-cell blocking values. Figure 6 shows a few examples of blocking patterns we used in painting with black ink. The second pattern from the top is drawn by hand and is mainly used in the results because of its natural appearance. Once the brush tip touches the canvas such a pattern is written into the blocking texture, locally "steering" the diffusion process in the next simulation steps. In practice, we combine values of the height field of the canvas with values from the blocking pattern to create a combined blocking value. This takes into account the fact that the canvas surface also affects the way pigment diffuses. The percentages in figure 6 indicate how much the pattern contributes to the total blocking value.

This procedure is able to mimic the blooming effect in the brush's footprint, as described in the previous section. In extending this technique to create similar

Figure 3: Applying a textured tissue to an area of wet black ink. The result is a print of the texture of the paper.

effects in a stroke once the brush is dragged on the canvas, we have to make sure the blocking pattern is applied at regular intervals. Moreover, the pattern has to disappear after some time period in order to avoid interferences with neighbouring strokes. This issue is handled by assigning each pattern in the blocking texture an age-value. Each time-step a small value is subtracted from the cells in the blocking texture, reducing them to zero over time.

The right image in figure 6 demonstrates some strokes drawn in black ink with the diffusion controller brush.

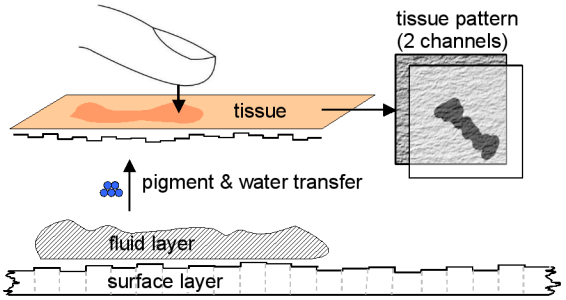


Figure 5: Removing water and pigment from the canvas surface with an absorbent tissue.

6.3 Masking fluid

The masking fluid forms a protective, solid layer on top of the canvas, securing all layers below. While painting with this special kind of pigment, instead of adding densities to the textures containing pigment data, we mark affected cells in a mask layer that has no role in the dynamics phase. This layer is drawn on top of the canvas.

Because the passive layers do not participate in the dynamics phase of the simulation, we only have to consider the effect of the masking fluid on the three active layers containing water quantities and pigment densities.

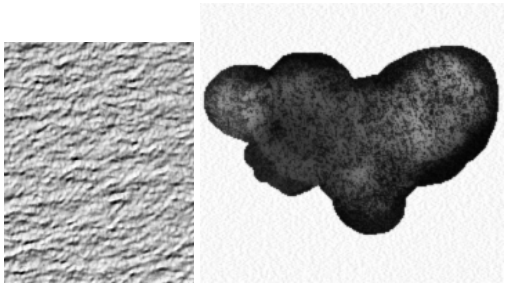


Figure 4: (*left*) A fragment of the textured tissue used in our system, rendered with per-pixel bump mapping. (*right*) A computer-generated example showing the resulting image after application of the textured tissue tool on a wet area of black ink. A distinct print of the tissue is still visible.

Similar to real masking fluid it is possible to cover it while painting; the paint just should not reach the canvas. This condition is assured by the ability to remove all pigment in the masked area when starting a new layer, or when removing the masking fluid.

When starting a new layer, all water quantities and pigment densities in the active layers covering the mask pigment are first removed, leaving the canvas untouched in that area. The same procedure is followed when removing the masking fluid.

The masking fluid is distinguished from strokes drawn with regular pigment by decorating it with whites crosses on red circles, as indicated in figure 8.

6.4 Selective eraser

A more versatile tool than the classic eraser that wipes paint from the canvas is the “selective eraser”. It can reduce or remove densities of a specific pigment from a mixture, both from the fluid layer and the surface layer. Similarly, it is possible to reduce or remove water quantities from the fluid layer and the capillary layer.

Because all simulation data is stored in float-texture objects, the implementation of this tool translates to a simple fragment shader that can be customized for the manipulation of one or more specific color channels of the data texture.

Despite its simplicity, the selective eraser has proofed it’s usefulness as an error-correction tool.

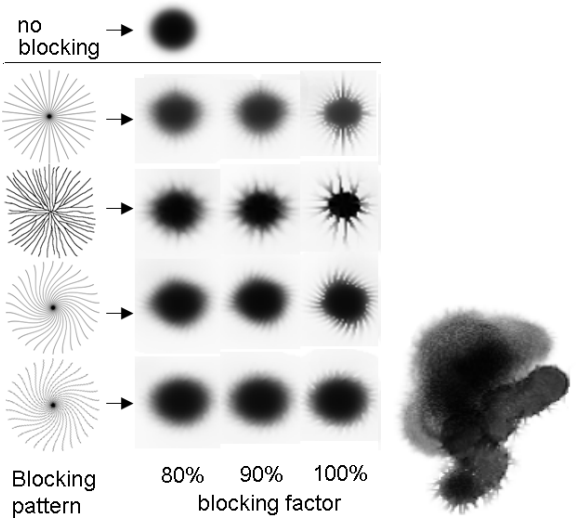


Figure 6: (*left*) Examples of blocking patterns used to "steer" pigment in the diffusion process. Each row shows the effect of 80%, 90% and 100% contribution of the pattern to the total blocking value. (*right*) A computer-generated example of some strokes drawn with the diffusion controller brush. The hand-drawn pattern was used to influence pigment diffusion, giving it its natural look.

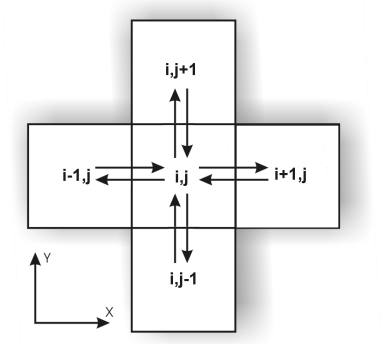


Figure 7: The unmodified diffusion algorithm makes cells exchange contents with neighbouring cells until an equilibrium is reached.

6.5 Paint profile

Our paint system is capable of simulating several related paint media: watercolor, Gouache and Oriental ink. Each one depends on a different setup of the simulation [VLVR05].

For example, Oriental ink requires a highly textured and absorbent canvas, and the dense carbon particles are so heavy they can be carried inside the canvas. Gouache on the other hand resembles watercolor but is applied in thick opaque layers on the canvas, creating flat color areas. Its palette contains a white pigment, which is not present in a watercolor palette. Therefore we have a large amount of settings that relate to the canvas, the fluid flow, the palette and the brush model.

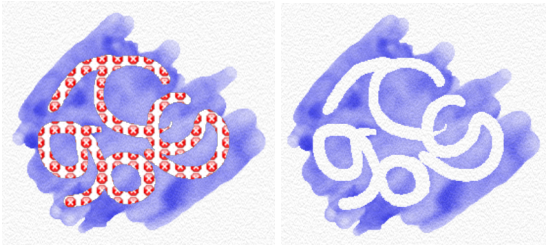


Figure 8: Using the masking fluid, which was previously applied by a brush (*left*), to cover the canvas beneath. To distinguish the masking fluid from regular pigment, it is decorated with a distinctive pattern. After removing the masking fluid the canvas remains untouched (*right*).

To cope with these different environments we introduce the notion of a paint profile, which combines all such properties and relates them to a specific name. The result is that we can easily change to a different paint media with a single mouse click and that we can even mix different styles in a single painting. At present our system supports three profiles: watercolor, Gouache and diffuse Oriental ink.

7 RESULTS

All computer-generated images presented in this work were drawn interactively and in real-time on a simulation grid of 512×384 cells, processed on a Intel(R) Pentium4 2.40 GHz system equipped with a NVIDIA GeForce FX 5900 graphics card. A Wacom tablet interface was used as an input device. As described in [VLVR05], a frame rate of 20 Hz is attained by processing only the areas of the canvas that correspond to "dirty" cells in an overlay grid.

The simulation results are displayed at higher resolution than the simulation grid by applying simple bilinear interpolation. Unfortunately, this also causes blurring of sharp stroke boundaries. One solution would be to increase the resolution of the simulation grid, at the cost of a slower simulation. A better approach is presented by Chu *et al.* [CT05], which consists of an image-based technique they call "boundary trimming". It restores sharp edges by reconstructing them with an implicit curve that is used to "trim" excessive pixels. This post-processing step could be integrated in our simulation to produce higher resolution images at low cost.

Figure 9, a painting in Oriental ink based on the work of Marie-Anne Bonneterre, demonstrates the use of both the textured tissue and the diffusion controller. The black background was taken from a real painting made with black ink.

The two results in figure 10 show how the masking fluid can be used to protect existing layers while painting a darker glaze on top.

Figure 9: A computer-generated image in Oriental ink, based on the original work of Marie-Anne Bonneterre. The distinct patterns on the flags are caused by usage of the textured tissue in combination with the diffusion controller. The black background was scanned from a real painting made with black ink.

Figure 10: Applying masking fluid to “oranges”, a watercolor work in progress (*left*), in order to place a message in a part of the orange that is darkened (*right*).

8 CONCLUSION AND FUTURE WORK

We augmented the capabilities of our paint system by means of several useful tools. Each one contributed to the expressiveness of the painting experience, facilitating the creation of images that lack the traditional artificial look of computer-generated images, as was shown in the results section.

There is more to a pigment than just the colour alone. Characteristics are e.g. the amount of staining, transparency, granulation, and spreading quantity. By using watercolor mediums, these extra characteristics can be modified. Future work includes investigating how we can offer these paint modifiers to the user efficiently.

9 ACKNOWLEDGMENTS

We gratefully express our gratitude to the European Fund for Regional Development (ERDF), the Flemish Government and the Flemish Interdisciplinary institute for Broadband Technology (IBBT), which are kindly funding part of the research reported in this paper. Part of the work is also funded by the European research project IST-2001-37116 ‘CUSTODIEV’.

We also would like to thank Marie-Anne Bonneterre and José Xavier for providing us with some real artwork and for sharing their expertise on the domain.

REFERENCES

[BSLM01] William Baxter, Vincent Scheib, Ming C. Lin, and Dinesh Manocha. dab: interactive haptic painting with 3d virtual brushes. In *Proceedings of ACM SIGGRAPH 2001*, pages 461–468. ACM Press, NY, USA, august 2001.

[CAS+97] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-generated watercolor. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 421–430. ACM Press/Addison-Wesley Publishing Co., NY, USA, 1997.

[CT05] Nelson S.-H. Chu and Chiew-Lan Tai. Moxi: real-time ink dispersion in absorbent paper. In *Proceedings of ACM SIGGRAPH 2005*, volume 24. ACM Press, NY, USA, july 2005.

[Mac05] Bruce MacEvoy. Wet in wet. Handprint: watercolour website, `http://www.handprint.com/HP/WCL/tech23a.html`, 2005.

[Smi98] Stan Smith. *The complete watercolour course*. Collins & Brown, second edition, 1998.

[Smi01] Alvy Ray Smith. Digital paint systems: An anecdotal and historical overview. *IEEE Annals of the History of Computing*, 23(2):4–30, 2001.

[Sta99] Jos Stam. Stable fluids. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 121–128, Los Angeles, 1999. Addison Wesley Longman.

[Sta03] Jos Stam. Real-time fluid dynamics for games. In *Proceedings of the Game Developer*, march 2003.

[VLVR05] Tom Van Laerhoven and Frank Van Reeth. Real-time simulation of watery paint. In *Journal of Computer Animation and Virtual Worlds (Special Issue CASA2005)*, volume 16. J. Wiley & Sons, october 2005.

Load Balancing on Cluster-Based Multi Projector Display Systems

Marcus Roth
Fraunhofer IGD Darmstadt
Fraunhofer Str. 5
64283 Darmstadt, Germany
marcus.roth@igd.fhg.de

Patrick Riess
Fraunhofer IGD Darmstadt
Fraunhofer Str. 5
64283 Darmstadt, Germany
patrick.riess@igd.fhg.de

Dirk Reiners
Human-Computer Interaction
Iowa State University
Ames, IA 50011, US
dreiners@iastate.edu

ABSTRACT

Sort-first or image space parallel rendering has been the subject of a significant amount of research. Its limited scalability is well known. Nevertheless, sort-first has noticeable advantages over other approaches: it has moderate and predictable communication overhead, and it has no problems with multipass or transparent rendering. Especially for multi-projector display systems, driven by a cluster of standard PCs with limited network bandwidth, it is the first choice to achieve interactive frame rates. In this paper we present an approach to do load balancing for arbitrary multi projector systems, based on a sort-first approach.

Keywords

multidisplay, cluster, sortfirst, realtime

1. INTRODUCTION

Many different multi projector display systems have been developed in the past two decades, mainly designed to overcome the restrictions of a single display. Multi projector displays like tiled walls provide a higher resolution, while systems like a Cave or a curved display allow a larger field of view. Nowadays, most of these systems are driven by a cluster of standard PCs. All of these displays are based on a static sort-first rendering, where each projector is responsible for a predefined part of the display. On a tiled display each projector is responsible for a rectangular region of the wall, and in a Cave a projector is assigned to one of the display walls.

Conference proceedings ISBN 80-86943-03-8 WSCG2006, January 30-February 3, 2006 Plzen, Czech Republic.

There are many different ways to construct a multi projector system, like having mono or stereo displays using active or passive stereo and with different amounts of overlap between tiles for edge blending. This paper describes how to configure a cluster based rendering system which can be used for arbitrary displays. As each projector has a predefined responsibility, the rendering performance is limited by the projector showing the most complex content. In the majority of cases the rendering load is not spread equally between the respective projections. As a result of this the performance can be enhanced if a PC which is connected to a display with low rendering load, absorbs a part of the load from a PC with high rendering load. To achieve this, a dynamic load balancing is necessary as load can change from frame to frame. The performance can be further enhanced if the number of parallel rendering PCs is independent of the number of projectors. This paper demonstrates how to utilize all the rendering performance of a cluster, where a subset of the PCs are connected to a projection system. Multi projector displays can be built in large scale. For example the *HEyeWall* of the Fraunhofer IGD in Darmstadt uses 48 projectors to build a stereoscopic tiled display with a resolution of 6144 x 3072 pixels. Currently there is no cluster network hardware

available to achieve a scene distribution (sort-last) parallel rendering on a display with 18 megapixels at interactive framerates. Additionally, current graphics cards are not able to render a part of the scene at full screen resolution, which is needed for direct sort-last rendering. As a consequence, the focus of this paper is on image distribution (sort-first) parallel rendering for load balancing on multi projector systems.

Load balancing can be split into two tasks: the estimation and the actual balancing of the load. The estimation needs to be both accurate and fast, which are contradictory goals. This paper presents a new estimation function that can be calculated efficiently, but also takes different aspects of the rendering load into account. This estimation function can be used to reduce the overlap factor that is responsible for the limited scalability of sort-first.

2. PREVIOUS WORK

A lot of work has been done for parallel rendering. In many cases algorithms were proposed to be implemented in hardware, but in the last few years an increasing number of algorithms have been developed that are optimized for cluster rendering. The classification of parallel rendering algorithms in sort-first, sort-middle and sort-last was proposed by Molnar et.al. [MCE94]. For software implementation only sort-first and sort-last approaches are usable.

With sort-first the screen is split into regions that are rendered in parallel, whereas sort-last splits the scene into independent parts to be processed simultaneously. In a cluster system with sort-first only the color of a pixel has to be transferred over a network to compose a final image. With sort-last the whole frame buffer with color and depth information of each rendering node has to be transferred for this purpose. Without the image composition step sort-last scales nearly linearly with the number of rendering nodes, as the rendering of scene parts is completely independent of each other. Sort-first does not scale very well because scene objects that are visible on more than one region have to be rendered multiple times. The influence of the overlap factor has been discussed in [MCE94].

To reduce the amount of data that needs to be transfered and to utilize the bandwidth and processing power of the cluster for sort-last a number of algorithms have been developed. One of the most popular ones is binary swap [MPH94], allowing interactive rendering on low resolution displays. Hybrid approaches have been proposed to achieve the scalability of sort-last as well as the low bandwidth requirement of sort-first [MRW99], but unfortunately with the combined advantages also the disadvantages are combined. Transparency and multipass rendering cause problems, and depending on the scene the amount of data that has to be transferred can reach that of binary swap. In [MWP01] a sort-last approach specialized for tiled displays is proposed, but the achieved framerates are less then one frame per second on a 4x4 tiled display. Although network bandwidth has been improved since the publication, it is not possible to get interactive framerates with current standard network hardware. As a result of this in most cases sort-first is used to do parallel rendering on tiled displays.

Samanta et.al. presented several approaches in [SZF99] to do load balancing on a tiled display based on sort-first. The proposed KD-split algorithm is also the basis for the balancing algorithm presented here. The projection area is recursively split into regions with equal load and then the resulting regions are assigned to rendering nodes. The approach presented here extends it to allow splitting of non-planar projections, and adds the influence of geometry and rasterization to the cost estimation function. Another problem with the original KD-split is that the screen split is done completely independent of the assignment of nodes to display regions. For this reason regions may be larger than the frame buffer of a single node, forcing them to be rendered sequentially in smaller parts.

[Abr04] solves the load estimation problem based on old frames' measured time and motion vector estimation. This enables a fast calculation independent of scene complexity, at the cost of limited accuracy for hard to predict motions like tracked head data. Even for scenes as complex as the powerplant the load estimation presented here has not been a noticeable factor, alleviating the need for less flexible and more complex load estimation.

3. DISPLAY CONFIGURATION

There are a lot of possibilities to construct a display consisting of a number of projectors and a number of PCs. To be able to handle as many different kinds of displays as possible the following situations must be handled:

- One PC can drive one or more projectors.
- Each projector can display a scene from an arbitrary view point.
- Each projector can display an arbitrary rectangular region of a planar projection.
- A projection can be mono, active stereo, or passive stereo.

- All situations can be combined.

One of the goals of the presented work is to support as wide a variety of display configurations as possible. Most systems are specialized for tiled displays, or non-planar displays and might have options for active or passive stereo. But the wide variety of possible configurations (e.g. tiled front wall with a single projector left side and a two projector floor) creates a combinatorial explosion of necessary specializations, especially when combined with the desire to support sort-first load balancing.

Therefore this work uses a more general way of describing a complete display. The core component is a Viewport. A Viewport has an associated camera and scene, and renders a single image. Each Window can contain an arbitrary number of Viewports.

This can be used to drive any kind of projection systems if we provide a convenient way to modify camera parameters. The core idea is to have one Camera, which holds parameters like position, orientation and view angle, which is shared by a number of Viewports to simplify changing parameters. To get left and right eye versions, or to split it up into tiles, a Camera can be manipulated using the Decorator pattern [Gam95]. Multiple Decorators can be combined to create compound modifications:

- **Tile camera decorator**: This decorator takes a viewing matrix and modifies it in a way that only a part of the viewing frustum is visible. For example on a tiled display this decorator is used to project a different part of an image with each projector.
- **Stereo camera decorator**: This decorator modifies the viewing matrix for the left eye or the right eye, depending on its settings.
- **Projection camera decorator**: This decorator modifies the camera so that it shows the scene from a viewpoint onto a viewing plane that is placed at an arbitrary position. In a cave this decorator describes the cave walls whereupon the position of the viewer comes from a tracking device.

With the decorators it is possible to modify camera parameters for all non active stereo displays. To be able to support active stereo the viewport is extended. With active stereo a PC has to render an image twice into different view buffers. To support this with Viewports, they can be dedicated to the left or right eye buffer, and for a complete stereo picture two Viewports are assigned to the same area of the Window, just to different eyes. Stereo parameters like eye separation are defined with the camera decorators related to a Viewport.

This structure allows handling of all displays which are composed of rectangular planar projection planes. To be able to support non-planar displays, like spherical projections, an additional display correction step was added that works on the frame buffer. After rendering is finished the frame buffer is copied into a texture (or on current systems rendering is done directly into a FrameBufferObject (FBO)) and it is projected onto a distortion geometry that can be defined for each projector. With the combination of all techniques it is possible to configure all displays that can be build out of standard projection systems.

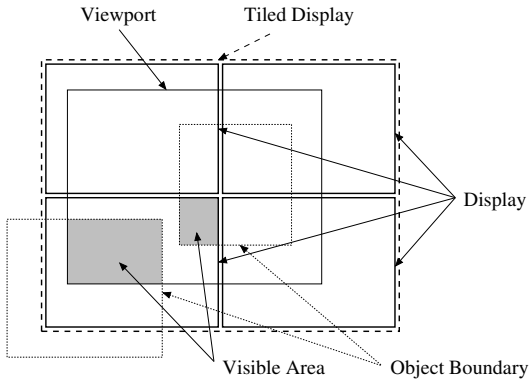


Figure 1: Virtual window

To extend this to parallel, clustered rendering a single, large virtual ClusterWindow is defined (see fig.1), which is split in two independent, different ways. One split is defined by an arbitrary number of Viewports. The second split is a distribution among all the nodes in the cluster associated with the window. These two splits are independent, i.e. one Viewport can be visible on many cluster nodes, or a single cluster node can have multiple Viewports. Each Viewport has a camera that defines which part of the scene is visible. To calculate the load for a single projector all viewports which are visible on the assigned area need to be found. To select the exact region that a cluster node is responsible for, a TileDecorator is added internally, which selects the parts of the Viewports that are relevant. This modified viewport is the basis for the load estimation. The load balancing assigns parts of one viewport from an overloaded PC to another.

4. LOAD ESTIMATION

The load estimation must be accurate and fast. As men-

tioned above it is not possible to achieve both goals at the same time as they are contrary. The best estimation result can be reached if the scene is rendered, because that definitely answers the question how long it takes to render it, but the resulting performance is worse than without balancing.

A common approach is to use rendering information from previous renderings. This can be very accurate, but has some disadvantages. With current hardware it is difficult to get exact rendering times because most of the rendering is done asynchronously. Another disadvantage is that rendering information from previous frames is only accurate if the viewpoint does not change very much. But this is a situation where low framerates are mostly not a big problem. If the viewpoint changes very fast, the performance values from previous frames are not accurate, but especailly in this situation high framerates are important. Based on these considerations interframe information is not used in this paper, and a fast load estimation based on the current camera settings is designed instead.

The load balancing proposed in this work is based on the scene graph system *OpenSG*, which is publicly available at *www.opensg.org*. The leaves of the graph represent geometric objects. In polygonal models those objects contain a number of geometric primitives. If a fast load estimation is desired, it is impossible to take each individual graphics primitive into account. But as a compromise information of a geometric object in the scene graph can be gathered, and used to build a load estimation for the whole object. Three different cost factors are extracted for the load estimation.

- **Constant costs**: These costs have to be calculated for each graphics object in the scene graph which is considered to be visible. Constant costs are independent of the fact whether the complete object is visible or just a part. They arise from the transformation of all vertices into the clipping coordinate system and the clipping of the primitives. Especially for a sort-first approach it is very important to take constant costs into account because they are incurred on each screen tile on which an object is visible. To estimate the number of transformed vertices to number of transfered indices is used.
- **Relative costs**: These costs have to be calculated for each visible primitive. As load estimation is not performed for single primitives, the whole geometry object has to be taken into account and its visible fraction calculated. This can be done by projecting the bounding volume into screen space and calculating the fraction that overlaps a viewport. Similar to constant costs the relative costs are related to the number of primitives in a geometric object. In the load estimation function again the number of indices is used as a fast indicator.
- **Pixel costs**: These kind of costs have to be calculated for each visible pixel of a geometry. As the number of visible pixels cannot be predicted exactly, it is estimated by projecting the bounding volume into screen space. The region overlapping the viewport is used as an estimation for the number of visible pixels. These costs mainly depend on material properties like textures and pixel shaders, they are not related to the number of primitives.

Each kind of cost is multiplied by a constant factor (C_c, C_r, C_p) that depends on the graphics hardware. For the material properties colored, textured, and materials with pixel shader are handled separately. For each of these different factors their respective cost is measured using an off-line program. This is a very rough approximation, but due to this simple approach it is possible to use fixed factors without the need to do a per object material analysis.

$$c(o) = i(o)\left(C_c + C_r\frac{a(o)}{r(o)}\right) + C_p a(o) \qquad (1)$$

The cost estimation function calculates the cost c for an object o. The function *i(o)* gives the number of indices used by an object, *a(o)* is the number of visible pixels of the projected bounding volume and *r(o)* is the size of the projected bounding volume into screen space without clipping. The cost estimation is very simple but it is able to cover the core parts of the rendering pipeline.

In a cluster system the performance of load estimation can be increased by doing it in parallel on all rendering nodes. When the load estimation is finished for each visible graphics object of the scene graph for all displays, the results are transferred over the network to a central node that is responsible for the load balancing.

5. LOAD BALANCING

Before starting to develop a new load balancing algorithm it is important to know the goal of the optimization. One possible goal would be that each parallel rendering node has the same rendering load. But especially with a sort-first based rendering approach this could lead to suboptimal rendering performance.

Fig. 2 shows a screen with two visible triangles. There are two possible ways of splitting this screen into two

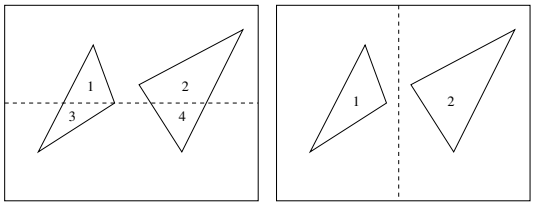


Figure 2: Possible screen subdivision with equal load

regions with equal load. On the left side both triangles are visible on both regions. So in each one, two triangles have to be processed. Summed over both regions, four triangles have to be rendered. In contrast to this on the right side only one triangle is visible on each region. So during rendering, only two triangles have to be rendered. This demonstrates that equal load is no guarantee for an optimal screen subdivision. As a result of this an equal load is not the goal in the first place, but instead searching for a screen subdivision where the overall rendering performance reaches a maximum. As the rendering performance depends on the rendering time of the region with the highest load, the screen subdivision where the region with the highest load reaches a minimum is selected.

In most approaches based on sort-first a region is subdivided along its longest side. This is often called the median cut [Whe85; Mue95]. This split is attractive because it minimzies the length of a region border, which is beneficial according to the Molnar/Eyles overlap function. In the described load estimation, this is already taken into account by the use of constant costs. This allows evaluating the benefits of cutting a region along both axes in order to compare the resulting load of both cuts, and selecting the one with the smaller maximum load. In contrast to the median cut this approach can be called the best cut.

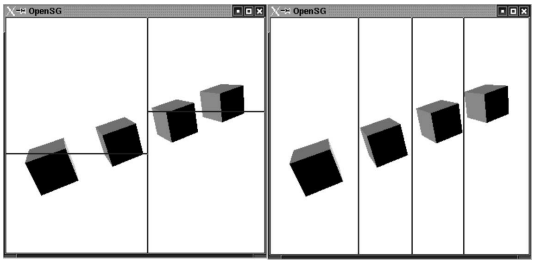


Figure 3: Median cut compared to best cut

From the load estimation a list of all visible viewports for each server as well as their sizes and positions are calculated. This includes a list of all objects whose bounding rectangles intersect a certain Viewport, and their constant, relative, and pixel costs.

To balance the load, each server's load is summed up. The servers are sorted by their load, and a median load is calculated. Load (i.e. a part of the screen) is reassigned from the highst ot the lowest load server until one of them reaches the median load. To avoid unnecessary pixel transfers a threshold is used, which limits splits of minimal effect. Only if the imbalance is greater than the threshold a transfer from one server to the other occurs. This procedure is repeated until each server in the table has a nearly equal load.

This simple load balancing is the basis for the following screen split algorithm. It is not generally possible to reach an exact balance as each screen split raises the overall costs according to the constant costs of the load estimation function.

To find the optimal split location for one axis (x or y) the objects that start or end at each pixel location on the axis are listed, and for each object the relative and pixel costs for each column resp. row are calculated. Using this structure as a basis the best cut can be found using the following algorithm:

```
c1 = 0
c2 = areaCost
r = 0
for cut = beginArea to endArea
    foreach o in openingList[cut]
        r  += o.relative + o.pixel
        c1 += o.fix
    c1 += r
    c2 -= r
    foreach o in closingList[cut]
        r  -= o.relative + o.pixel
        c2 -= o.fix
    maxC = max(c1,c2)
    if(c1 ge c2)
        break
    lastMaxC = maxC
    lastC1 = c1
    lastC2 = c2
if(lastMaxC lt maxC)
    return (cut-1,lastC1,lastC2)
else
    return (cut,c1,c2)
```

This is done for both x and y, and the best cut that can be rendered in minimum time is chosen.

If a region is split into two areas, where one area falls into the visible projection area of server one and the other region is calculated by server two, then the result of server two has to be transferred over the network to combine the correct image. Network transfer can be

minimized if the largest of both areas falls into the visible projection area of server one. To achieve this the center of geometric complexity for each area is calculated. As complex geometry results in smaller areas due to the load balancing, the least number of pixels have to be transferred if the center of complexity is calculated by the server which has to transfer its pixels.

The result of this procedure is a work list for all servers that is transferred over the network. To minimize network overhead this list is distributed using reliable multicast, which is provided by the OpenSG network layer.

6. RENDERING / PIXEL TRANSFER

After the work list is distributed, each server at first renders all screen areas which have to be sent over the network. After rendering, they are stored in main memory. In the next step the area that falls into the responsibility of the server is rendered. Areas are not immediately sent over the network because in an ideal situation all servers take the same time to process all the work in the work list. So it is best to let all servers do their work, and then work on the data transfer because all servers are ready to send and receive.

Pixel data is sent as 24 bit RGB color values without compression. Some forms of image compression have been evaluated, but most compression algorithms take more processing time than is gained by the reductioning transfer. A simple form of compression would be to use 16 bit color values, but it would result in noticeable quality loss, which was not deemed acceptable.

One problem with sort-first is that many rendering nodes often have to send data to a single node. If all nodes have the same network bandwidth, this could easily overload the receiving capacity of the single node. If socket communication over ethernet is used, an overload of a connection can cause the protocol to do balancing operations to avoid thrashing. These operations can cause a communication channel to be stalled for a large fraction of a second, pushing the resulting network throughput far below the optimum receiving capacity. To avoid network overload a demand driven network transfer is used. A receiving node requests a transmission from a sending node at the point of time when it is ready to receive. Multiple areas are being sent sequentially with the full receiving bandwidth without causing network overload. This can be optimized further if the request for one area is sent shortly before the transmission of the last area has finished. This optimization effectively avoids delays between the transmission of multiple areas.

7. SINGLE-SCREEN DISPLAY RESULTS

The load balancing algorithm was tested with three scenes which have different characteristics. The first is the dragon model with 8 million polygons, which is split into 1024 pieces in the scene graph. Rendering of this model is completely bounded by its geometry, and the faces are evenly distributed over the surface of the dragon. The second model is the power plant, containing 13 million polygons. An interesting feature of this model is that most of the geometry is located in a very small area of the scene, which makes it very difficult to get a good speedup with sort-first approaches. The third model is a BMW 6 series with 4 million polygons, where the coat is rendered with a complex pixel shader. Rendering of this model is limited by the pixel fill rate.

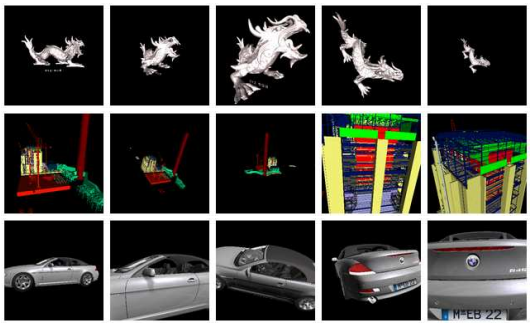


Figure 4: Test models

For all three scenes an animation path that contains a closeup and an overview camera position was defined. In all tests this animation was used to produce performance values. Images of the animation are shown in fig. 4.

As discussed above the main goal was to develop a load balancing for multi projector display systems. To evaluate the balancing results the results of parallel rendering on a single display with a resolution of 1024 x 768 are shown in fig. 5.

The framerate is calculated as the average of all animation images. With all scenes a significant speedup can be noticed. Using four PCs a speedup of at least two is reached. The speedup strongly depends on the rendered scene, with the dragon model reaching the best speedup. The reason for this is that this model is split into 1024 equal sized non-overlapping objects. The BMW 6 model is rendered unmodified. It contains very large and very small geometry objects that are not evenly distributed over the screen.

The cost estimation function contains fixed, relative and

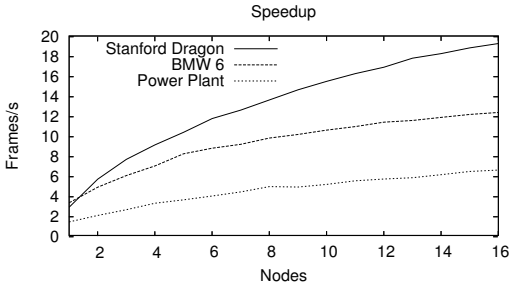


Figure 5: Models displayed on a single screen display with a resolution of 1024 x 768

pixel costs. Now the question is how these three factors are affecting the balancing result. Thus the scenes are rendered three times always using only one factor and one time using them combined. The results can be seen in fig. 6. They show that the deciding factor depends on the type of the model, making the use of the full cost function most effective.

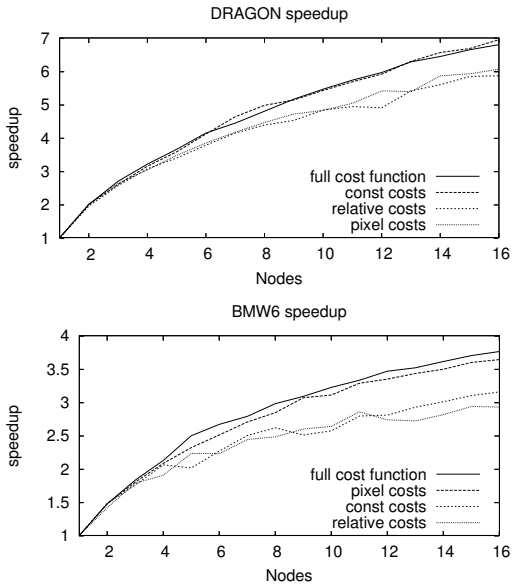


Figure 6: Influence of the cost estimation function for the BMW and Dragon model

8. MULTI-SCREEN DISPLAY RESULTS

As an example of multi-screen display system a tiled display with 24 projectors and a resolution of 6144 x 3073 pixel was used. For each scene the rendering time without load balancing is given. Generally the rendering on a tiled display is faster than rendering on a single display, because each projector shows only a part of the whole scene. The second case shows the same rendering with load balancing. For this test only the 24 PCs, connected to the display wall, are used for rendering. In a third test 24 additional PCs in the same cluster are used to check if further speedup is possible.

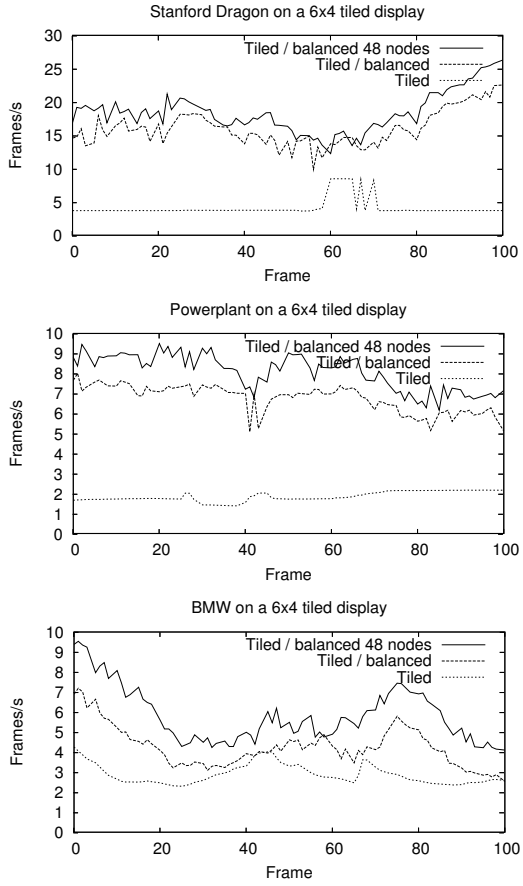


Figure 7: Three scenes rendered on a 6 x 4 tiled display

Fig. 7 shows the frame rate for each frame of an animation with 100 frames. Using the new load balancing algorithm it is possible to speed up the rendering on a tiled display significantly. It is possible to get a further speedup by using additional PCs.

In fig. 8 the time for load estimation, load balancing, buffer I/O, and rendering is shown. To generate this image the Dragon model was rendered on a 2 x 2 display wall with four display PCs and 12 additional render PCs. The time is given in percent of the whole frame time.

The frame time is mostly dominated by rendering. The time for load estimation and load balancing is small

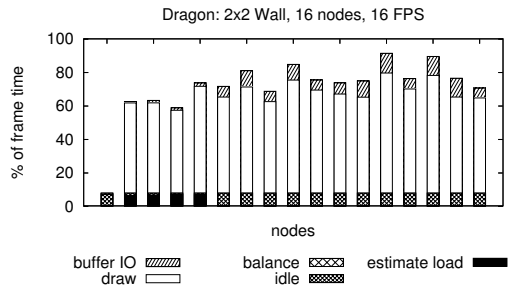


Figure 8: Dragon on a 2 x 2 Wall rendered with 16 PCs

compared to the whole frame time. The graph is not showing any values for network communication, but communication is part of the 100 percent frame time. It is not possible to show the real network overhead as a correct time value for each node. The reason for this is that in a communication the time for sending and receiving always depends on two nodes. However, it is easy to get a feeling on how network communication affects the rendering performance. The gap between the node with the most load and the 100 percent mark is mainly caused by network transmissions.

The results show that load estimation and load balancing are very fast. The balancing results depend on the scene. With the Dragon model, a similar rendering time can be achieved for all nodes. Different rendering times between the rendering nodes are a result of the simplifications that have been done during the load estimation.

9. CONCLUSION

A cluster based parallel rendering algorithm with sort-first based load balancing for arbitrary multi display projection systems has been presented. It uses a new load estimation function that incorporates costs caused by geometries overlapping more than one screen region. This cost estimation function enables using a *best cut* approach in contrast to *median cut*, which has been used in most other approaches. The results show that the reachable speedup strongly depends on the scene. But even scenes with a very uneven complexity distribution like the Power Plant can be rendered with a significant speedup. All the presented algorithms are part of the scene graph system *OpenSG* that is publicly available at *www.opensg.org*.

10. ACKNOWLEDGEMENTS

The dragon model was obtained from the Stanford Scanning Repository, thanks for providing it. The power plant was obtained from the Walkthrough project of the University of North Carolina, thanks to them and their anonymous donor for the model. The BMW model was provided by BMW, thanks for allowing us to use it in this publication.

11. REFERENCES

[Abr04] Abraham, F., Celes, W., Cerqueira, R., and Campos, J., A load-balancing strategy for sort-first distributed rendering, Proceedings. 17th Brazilian Symposium on Computer Graphics and Image Processing, pp. 292–299, 2004

[Gam95] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995

[MPH94] Ma, K.-L., Painter, J. S., Hansen, C. D., and Krogh, M. F., Parallel volume rendering using binary-swap compositing, IEEE Computer Graphics and Applications, 14, no. 4, pp. 59–68, 1994

[MRW99] Mckinley, P., Rao, R. T., and Wright, R. F., H-RMC: A Hybrid Reliable Multicast Protocol for the Linux Kernel, Proceedings of IEEE SC99: High Performance Networking and Computing, 1999

[MCE94] Molnar, S., Cox, M., Ellsworth, D., and Fuchs, H., A Sorting Classification of Parallel Rendering, IEEE Computer Graphics and Applications, pp. 23–31, 1994

[MWP01] Moreland, K., Wylie, B., and Pavlakos, C., Sort-last parallel rendering for viewing extremely large data sets on tile displays, IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, pp. 85–154, 2001

[Mue95] Mueller, C., The Sort-First Architecture for High-Performance Graphics, Symposium on Interactive 3D Graphics (Monterey, California, April 9-12, 1995), pp. 75–84, 1995

[SZF99] Samanta, R., Zheng, J., Funkhouser, T., Li, K., Jaswinder, and Singh, P., Load balancing for multi-projector rendering systems, 1999 SIGGRAPH / Eurographics Workshop on Graphics Hardware, pp. 107–116, 1999

[Whe85] Whelan, D. S., A multiprocessor Architecture for real-time computer animation, Ph.D. thesis, California institute for Technology, 1984

Agents Based Visualization and Strategies

Nicolas Roard
Swansea University
csnicolas@swansea.ac.uk

Mark W. Jones
Swansea University
m.w.jones@swansea.ac.uk

ABSTRACT

This paper describes a flexible visualization architecture based on software agents, which enables the abstraction and reuse of rendering strategies. Using a reification of the rendering environment, the system is able to add new rendering strategies (such as distributed rendering or progressive rendering) to an existing pipeline, without any modification of the other components (controls components, display components, rendering algorithms, *etc.*). The ability of changing strategies on the fly leads to a better adaptability to runtime constraints. The system uses an agent-based graphic pipeline, where each agent/component can be located on different computers; communications between agents use XML/RPC and data stream in order to easily integrate existing code in the system. Agents can add specific behavior to graphic pipelines, such as saving environments to reuse them, adapt information and knowledge from another pipeline, and generally modify and improve the entire system. Various visualization and control clients exist, enabling collaboration between platforms such as PDAs, Windows, Linux, MacOS X, and Web (using Java applets).

Keywords Distributed Visualization, Software Agents, Volume Rendering

1 INTRODUCTION

Although individual graphical capacities continually improve on workstation or desktop computers, visualization at interactive frame rates is still a problem with very large datasets or complex rendering algorithms. This is particularly evident in scientific visualization, such as medical data or simulation of fluid dynamics; high-performance computing facilities organized in a distributed infrastructure need to be used to achieve reasonable rendering times in those cases [HEvLRS03, BBC+05].

Such distributed visualization systems are required to be more and more flexible; they need to be able to integrate heterogeneous hardware (both for rendering and display), span through different networks or the internet, and easily reuse existing software. They also need to allow complex user interactions like collaboration [MF00], and easy customization to answer specific needs.

Complex distributed software systems tend to be hard to administrate, and tend to respond poorly to faults (hardware or software). The Autonomous Computing grand challenge initiated by IBM [KC03, IBM] aims to build software systems that are as autonomous as possible to simplify implementation and administration.

WSCG 2006 conference proceedings, ISBN 80-86943-03-08
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

Our system is a reflective middleware [KCBC02] based on agents, which we used to implement a distributed graphic pipeline. We worked on a Volume Rendering pipeline, as Volume Datasets by nature tend to be both large and slow to render, and are thus good candidates for testing a distributed visualization system.

We will first review related systems and present the general architecture of our system. We will then present some examples showing the reflective nature of the system, first by talking about the Visualization Strategy pattern, then by introducing added behavior to generic pipelines.

2 RELATED WORK

Sharing computing resources is an old idea in computer science, but the advance of Internet and Web Services now permit the building of interoperable, cross-platform distributed services. The Grid Initiative [FK99, FKT01] and projects like the Globus Toolkit [FK97, Fos05] provide toolkits and infrastructure to deploy such grid computing systems.

Visualization systems using the grid have their own set of requirements [SWB03, BBC+05], and are an ongoing research subject (see [ADK+99, BDG+04, KHRV04, GAW05, RWB+05] for some of the existing visualization systems using the grid). These systems implement a distributed graphic pipeline following the established modular dataflow model [UFJK+89, Dye90, Cam95] using grid technologies.

Multi-Agent Systems (MAS) are a particular architecture of distributed systems. They tend to be more flexible and reliable than traditional distributed systems, as Software Agents can work and react at a local level, providing a decentralized intelligence. As such, they

seem an interesting research idea to obtain autonomic systems, and more reliable and flexible distributed systems [JJZ+04, ZM05, GFB05, MGR+99].

Some research projects follow an interesting approach, combining an agent system with a distributed rendering system [HEvLRS03, RKACM03].We followed a similar direction, although our orientation is slightly different than those systems. We wanted a flexible research tool, with reflective capabilities [KCBC02, ASA01], to experiment how we can use thoses capabilities to build intelligent applications.

3 GOALS AND PHILOSOPHY

Our principal goal was to obtain a flexible system for experimenting with ideas and architectures. We believe that using loosely coupled, dynamic systems and languages, leads to simpler and more powerful systems, which are easier to prototype.

The design thus focused on building a layered system (Figure 1), with each layer kept simple.

7	Autonomic System
6	Intelligent Applications
5	Reflective System
4	Graphic Pipeline
3	Agents
2	Distributed System
1	Processes

Figure 1: System layers

Remote Processes (1) interact in a Distributed System (2) by sending messages. On top of this Distributed System we build an Agent infrastructure (3) – in fact we consider everything to be an agent in our system, which gives the next layers a range of useful functionalities (creation, discovery, *etc.*). Using Agents, we then implemented a graphic pipeline (4). Agents are aware of their environment and can modify it, in essence creating a Reflective System (5). The agents environment comprises not only the graphic pipeline, but also the complete system. We then take advantage of this Reflective System and the autonomic nature of Agents to build Intelligent Applications (6). Our final objective is to build an Autonomic Visualization System based on the existing layers (7).

Our current interest and what we describe in this paper is in exploring the Intelligent Applications layer – how can we take advantage of the Reflective nature of the system to build applications or interesting architectures.

4 SYSTEM ARCHITECTURE

4.1 Core System

The core system is a very simple distributed architecture, composed of a naming service and agents, where agents communicate using XML/RPC [Win99]. Each agent is associated to a quintuplet *{location, port, name, type, status}*. Ports can be shared among agents.

Using XML/RPC gives us a simple solution for sending remote messages, which uses very common standards (TCP/IP, XML, HTTP). An immediate benefit of this simplicity is the fact that most mainstream computing languages have a working implementation of XML/RPC (*e.g.* [Mül03, ASF05]). It is therefore a simple task to transform existing code into a component of our system.

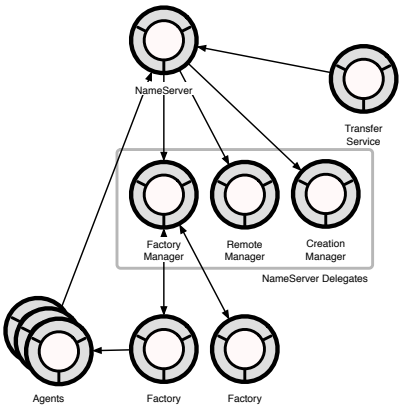


Figure 2: Architecture overview

An agent can be duplicated easily or moved to another machine, which gives us a very flexible network topology. Furthermore, the system is very dynamic and reflexive, as agents can modify the system at runtime, request information or meta-information about the system or the graphic pipeline, and add new information to the system.

Figure 2 shows an overview of the basic architecture. On each machine we have a NameServer which keeps track of all the objects on the local machine and of remote NameServers. A Transfer Service agent can be used by agents to move data from one machine to another. Factories (Agents that create other agents) are handled via a Factory Manager. The Remote Manager agent is used to manage a network of machines and forward requests. The Creation Manager is in charge of managing the population of agents on the available machines. The following sections detail each component.

4.2 Naming Service

A new agent which wants to register with the system contacts the local Naming Service (using a default port). The Naming Service then returns an available local port to the agent. The agent can then initialize and finish its registration.

4.2.1 Extending the Naming Service

Additional agents can register to the local Naming Service as delegates, providing a Naming Service themselves. Then, in case of an unsuccessful request to the local Naming Service, these agents are called and can answer the request (see Sections 4.4.1 and 4.5). In our current systems those delegates are the Factory Manager, the Remote Manager and the Creation Manager.

4.3 Transfer Service

The Transfer service is an agent providing simple transfer methods, able to send data/binaries from a machine to another. It is needed by the Replication mechanism as well as Data management.

4.4 Agents

Agents are autonomous software that can interact with their environment. In our system, we also need to be able to replicate agents easily and create them on demand.

4.4.1 Agents Creation: Factories

Factories create new agents of a certain type on demand. Factories are implemented as normal agents answering a certain protocol (a protocol beeing a set of messages) with the type FACTORY. A Factory Manager agent registers as a delegate to the NameServer. Each Factory is automatically registered to it.

When an agent is requested, the NameServer searches to see if an available instance exists in the system. If not, the request is delegated to the Factory Manager, which in turn calls the corresponding Factory and asks it to create a new agent. This new instance is then registered to the NameServer and returned.

A Factory cannot create an agent if the system is overloaded (in which case other machines can possibly handle the creation).

4.4.2 Replication

The process of replication of an agent is twofold; first, we consider the agent binary (or sourcecode if the agent is a script). Second, we also need to replicate the agent's state if we want to move an agent rather than create an agent of the same type. Replication helps with the autonomic aspect of the system – i.e. self repair.

In our current system the agent's state is determined by the pipeline and the environment (see Section 4.6), so we do not need to manage the serialization of the agent's state. We only need to register the new agent to the pipeline, and restart it. The start method in the agent will use the pipeline to initialize itself.

4.5 Managing Multiple Machines

Using Factories and Replication, we can have a remote creation process to use multiple machines in our architecture. The Creation Manager agent use the Transfer & Replication services to create new Factories on available machines, depending on their state (*e.g.* CPU load, disk space).

The naming mechanism is extended via a NameServer delegate (similar to the Factory Manager), called the Remote Manager. Machines are organized in a hierarchical way (Figure 3) where each child machine registers to the Remote Manager.

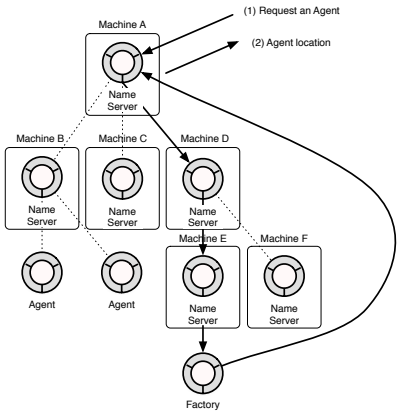


Figure 3: Remote creation

When the NameServer gets a request for an available agent, and none is found locally, it forwards the request to its delegates. If a delegate doesn't return an agent's address, the next delegate is called. Figure 3 shows this mechanism.

For example, if the Factory Manager doesn't return an agent (because it doesn't have a corresponding factory, or because the maximum number of agents is reached, or the CPU load is too high), the Remote Manager is called and try to answer the request. The Remote Manager maintains a list of the machine's children, and a cache of their factories types. It then forwards the request to a child having the corresponding factory. If the Remote Manager itself can't handle the request (*e.g.* no corresponding factory) the Creation Manager is called and using the Transfer & Replication service can create new Factories.

4.6 Visualization Architecture

We implemented a distributed graphic pipeline using our system, where each pipeline component is an agent. Figure 4 shows the general structure of a pipeline.

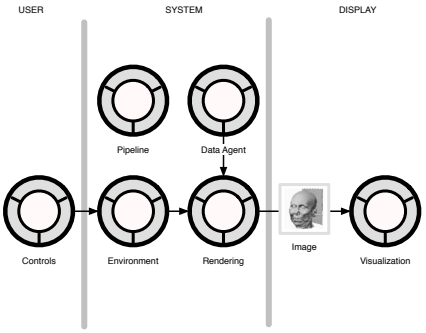


Figure 4: Visualization System Architecture

A Rendering agent gets its data information from a Data agent, and uses a "rendering environment" given by the Environment agent to render an image. The image is then sent to a Visualization client.

The Environment contains all the information for the rendering – quality, specific settings, camera position,

etc. – and can be modified by other agents, like a Control agent to move the camera.

The Visualization and the Controls are in general simple clients to the system, not real agents, and thus can be completely externalized, which permit for example to run them through a web page (using applets), or in a client application (Section 4.8).

Other agents in the pipeline need to be fully integrated (shown on Figure 4 by the gray outline) in the system. Agents in a pipeline are registered in a Pipeline agent and can be reached and manipulated by other agents through it.

Agents communicate by sending XML/RPC messages (synchronous and asynchronous) and direct socket transfer for transmitting data. The overhead of XML/RPC is not a problem in general, specifically by using asynchronous messages whenever it's possible[1] and limiting the number of necessary messages in the architecture.

4.7 Data Management

Data needs to be duplicated and moved along with the rendering components of the pipeline. Data agents are in charge of that task, and use the transfer facilities to move or duplicate a dataset to another machine. In the future we envision intelligent agents and data placements using planning agents, but for the moment the data management simply transfer data from the original pipeline to cache it locally.

4.8 Clients and User Interface

As explained in Section 4.1, XML/RPC permitted a quick development of the system and an easy integration of existing components. A good example is how we were able to program visualization and control clients for various platforms, taking advantage of specific languages/frameworks allowing quick prototyping. We used GNUstep [FSF] and Cocoa [App] frameworks to program a crossplatform Objective-C/OpenGL client running on Windows, MacOS X and Linux. Squeak [Squ] (a Smalltalk environment) was used to prototype the system and provides a PDA implementation, and Java was used for the web client. Figure 5 shows the PDA and Java clients, while Figure 6 shows the Squeak environment used for prototyping. Figure 7 shows the GNUstep client.

In parallel to this project, we participate in another project which will integrate this agent system using the grid toolkit in order to access the security framework provided by that system.

4.8.1 Modifying the User Interface

One problem with adding unplanned, "intelligent" functionalities to a graphic pipeline as proposed in Section 7 is obviously that the user interface needs to be changed

[1] while XML/RPC doesn't specify asynchronous messages, many implementations provide them, as it is a very simple modification

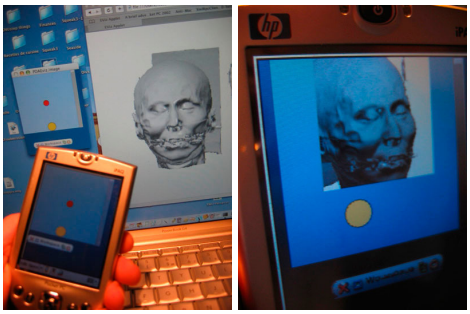


Figure 5: The PDA and Web clients, with the PDA acting as a remote control on the left, and as a visualization and control client on the right

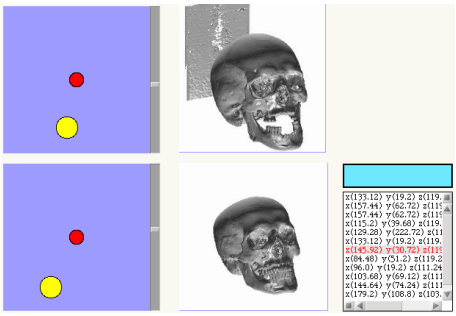


Figure 6: The Squeak User Interface with two different datasets, linked by a mediator agent

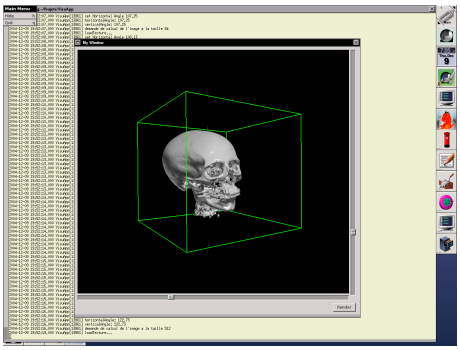


Figure 7: GNUstep visualization client on Linux

to use them. At the moment we mostly use a Squeak UI for rapid prototyping (Figure 6 shows the mediator agent along with the viewpoint agent described in Section 7) which make it easy to add new elements to the user interface, but we plan to extend our current web client to automatically generate the user interface for a pipeline (currently the web interface uses two java applets, one for visualizing the results of a pipeline, and one for controling the point of view), as html combined with java applets would gives us a very customizable and extensible user interface.

5 APPLICATIONS OF THE SYSTEM

The previous sections detailed the general system architecture. We will now introduce some applications of the reflective nature of the system.

The graphic pipeline is reificated through different agents; agents can use this architecture to gather information,

or even modify the architecture. The following sections show some examples of this approach:

- Section 6 “Visualization Strategies” details the Visualization Strategy pattern and its applications.
- Section 7 “Modifying the pipeline” introduce examples customizing the pipeline with additional agents adding new functionalities.

6 VISUALIZATION STRATEGIES

6.1 Presentation

If we look at the architecture for a standard rendering loop, Figure 4 can be simplified as shown in Figure 8.

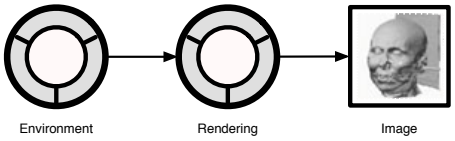


Figure 8: Simplification of the architecture

In this architecture, we consider a Rendering process as a Rendering agent using an Environment to generate an Image.

6.1.1 Visualization Strategies

A Visualization Strategy is a specific visualization process, like splitting the viewing output in different images, or distributing a rendering. We encapsulate such visualization patterns into a single agent. This agent answers the same rendering protocol as a “real” rendering agent, and can thus be used without any modification in place of a rendering agent – the other parts of the pipeline (*e.g.* the visualization client or the controls in Figure 4) are left untouched.

6.1.2 Genericity and Composition of Strategies

In general, Strategies do not provide a rendering algorithm themselves, but use existing rendering agents implementing the algorithms.

As long as an agent answers the rendering protocol it can be used transparently in a Strategy; which means that, for a given Strategy, any of the available rendering algorithms can be used. Conversely, we can say that Strategies are generic behaviors: a new rendering algorithm will be able to transparently take advantage of the existing strategies in the system.

As Strategies respond to the rendering protocol, they are considered by the system as Rendering Agents, and can thus be composited: in a given Strategy, instead of a “real” Rendering Agent, another Strategy can be used.

One needs to take care of the composition cost when building complex composited pipelines as performance can be impacted by the communication overhead.

The following sections demonstrate some of the strategies we created.

6.2 Progressive Rendering Strategy

Progressive rendering is a mechanism that computes a rendering in a low resolution, then gradually increments the resolution to improve the quality. Figure 9 shows an example of a 3-step progressive rendering for a volume dataset.

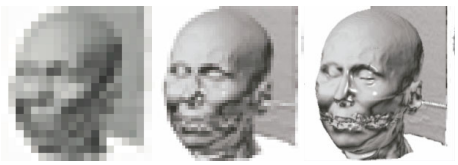


Figure 9: Progressive rendering

Although progressive rendering is a longer process to get the final image than calculating the final image directly (unless it is distributed by the pipeline), it’s a very useful mechanism, as it allows the user to have a quick feedback of what will be the final result. Moreover, the low quality rendering can be fast enough to achieve interactive frame rates.

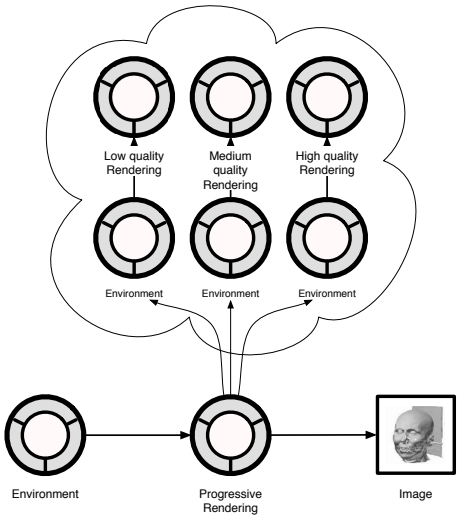


Figure 10: Using a strategy: progressive rendering

To create a progressive rendering strategy, we use an agent answering the rendering protocol, which will get the environment and where visualization clients can connect to.

This agent implements a three-step progressive rendering, and so requests three rendering agents for its needs. When receiving a rendering request, it modifies the resolution in the rendering environment, and then passes the new environment to the rendering agents. The progressive rendering agent is registered as a *visualization client* to each of the rendering agent. When it gets a result, it forwards it to the “real” visualization client.

Figure 10 shows the architecture of this progressive rendering agent – as can be seen on this diagram, the only modification lies “in” the rendering agent, and consists of only modifying the environment. All the other parts

of the pipeline are left untouched, an important characteristic of the Visualization Strategy pattern.

6.3 Distributed Rendering Strategies

The general approach to Distributed Rendering consists of splitting the computation load of a rendering on multiple machines, creating several partial results, then merge them to get the final result.

We implemented two approaches to distribute a rendering process, by using image-space methods or object-space methods to split the computation load, then merge the rendered isolated fragments in a single image as a final step.

The general idea is that once you have a process that can adequately split the rendering over multiple agents and then merge the result, the agents can easily be scattered on different machines, and their distribution controlled by another agent depending on specific parameters (*e.g.* taking in account the load of the machines).

6.3.1 Image-Space Splitting

We consider here the rendered image as our work space. We want to split it in multiple parts, where each part of the final image is rendered by a separate agent.

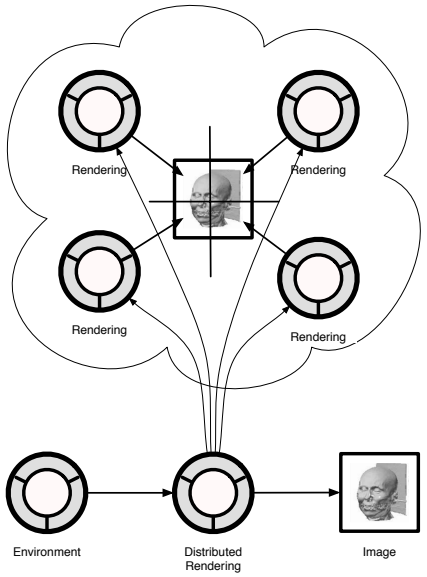


Figure 11: Image-space distributed rendering

To do that, we simply need to compute four different camera viewpoints that match the desired parts of the image. We do that in a strategy agent that gets the original rendering environment and extracts the camera viewpoint from it. It then sends corresponding computed viewpoints and "look-at" points to the four different rendering agents it requested (Figure 11 shows the architecture).

6.3.2 Compositing of Image-Space Parts

Merging the resulting partial images is very simple in that case, as we only need to build the final image by aggregating the different parts at their correct position.

6.3.3 Object-Space Splitting

With Object-Space Splitting, the idea is to render only a part of the dataset instead of the complete dataset in each agent. The dataset can be either physically split into multiple parts and each part sent to rendering agents, or alternatively have a rendering algorithm that accepts to render a part of a dataset (the right approach depends on the size of the dataset; we only implemented the selective part rendering for now).

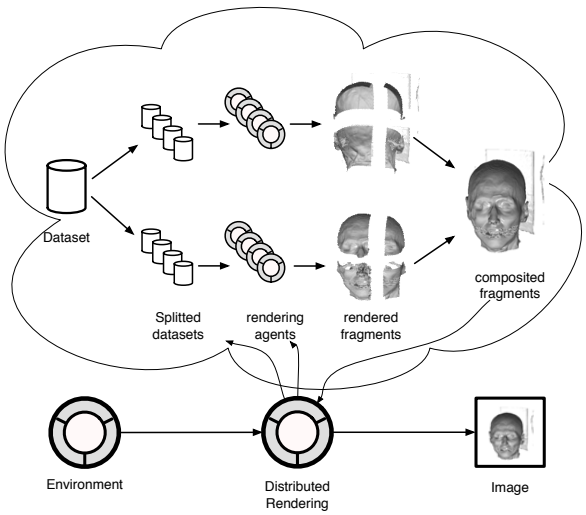


Figure 12: Object-space distributed rendering

Figure 12 shows the general architecture. A rendering strategy agent is used in a pipeline. This agent then computes the different data segments and sends them to the rendering agents. Each rendering agent will render a partial dataset into an image. Renderers need to generate the alpha channel for the image.

6.3.4 Compositing of Object-Space Parts

We obtain the final image by merging the partial images in the rendering strategy agent, using a simple back-to-front rendering algorithm.

7 MODIFYING THE PIPELINE

One of the differentiating aspects of our system is its reflective nature – agents can inspect the system and modify it. The following sections detail some examples of how agents can be added to a graphic pipeline to add functionalities.

7.1 Framerate Steering

As a first example, we would like to have a pipeline where the resolution of the image is tied to the framerate; that is, automatically set the resolution to match the desired framerate as close as possible.

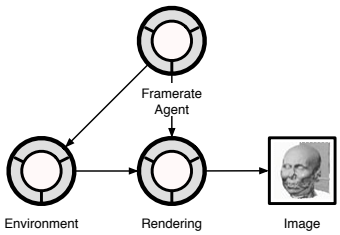


Figure 13: Framerate steering agent

We can do that by adding a single agent to a standard pipeline (Figure 13). The agent registers as an observer to the rendering agent, to monitor the rendering times. This framerate agent can then create a feedback loop by modifying the image size in the pipeline environment until the minimum framerate is reached.

This simple behavior could be extended by choosing among different rendering agents instead of fixing the image size.

7.2 Saving Viewpoints

In a graphic pipeline, the immediate agents' environment is the pipeline itself and the associated agents (see Figure 4).

One application we developed is a method to save the current camera position (viewpoint) and retrieve a list of the saved positions, so the user can build its own customized list of usual positions for a particular dataset.

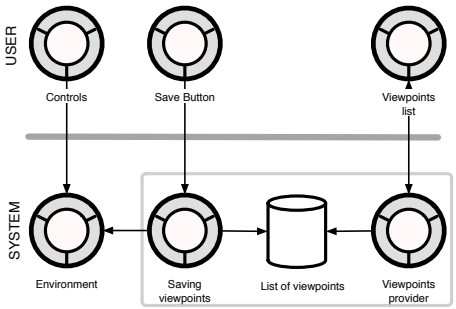


Figure 14: Saving and using viewpoints

Figure 14 shows the architecture enabling this functionality. We add three agents (framed on the figure) on the system side, one holding the viewpoints list (a data agent), one that can add a viewpoint to the list by checking the current viewpoint in the pipeline environment, and the third one that can returns the list of viewpoints. On the user side, we need two clients, one to request the addition of the current viewpoint, one to request the list of viewpoints.

This system can be used in any pipeline and, albeit it's a simple example, highlights how agents can add new knowledge to a pipeline and how they can use it.

7.3 A Mediator Agent

Another example of how agents can interact with a pipeline and extend its functionalities is a mediator agent, which can transform automatically some coordinates (*e.g.* camera position) used to visualize one volumetric dataset into equivalent coordinates for another volumetric dataset.

We can then manipulate one pipeline and have the movements replicated on a second pipeline, in real time, after transformation.

The current prototype can be used as the start of a collaboration mechanism, and we also want to use it in the future coupled with the "saving viewpoints" agents to automatically use viewpoints saved for one dataset with another, which can be useful for medical applications.

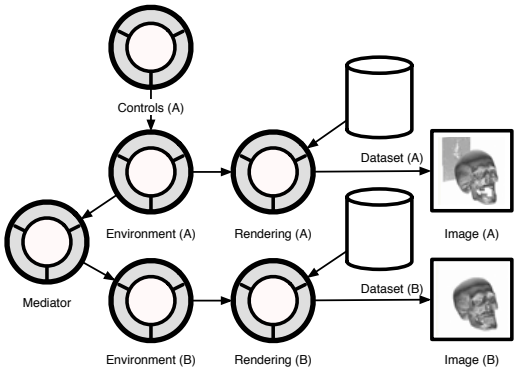


Figure 15: A mediator agent

Figure 15 shows the actual architecture of the system. We use here two pipelines running in parallel, with a mediator agent coupling the pipelines' environments; the mediator agent is registered as a listener to the environment agent of the *Environment A* and knows the deltas between both environments (after a calibration step). Using the controls of *Pipeline A* will update *Environment A*; the Mediator is then notified of the change, and will update *Environment B* (see Figure 6 for a screenshot of the current system).

8 PERFORMANCES

For the 256^3 data sets rendered in these examples, all these agents work in real-time, and achieve high frame rates when distributed (> 25 fps). We aim in the future to use UDP instead of the current TCP based transmission mechanism in order to augment the performances and improve the latency of the system, particularly when dealing with bigger datasets.

9 CONCLUSION

The use of web services and XML/RPC gave us the opportunity of mixing different languages and programs in a common system, allowing us to take advantage of each language's/platform's strong points.

Focusing on a simple, dynamic, agents architecture and building our system on top of it proved to be very useful during the conception of the architecture. This dynamism also allowed us to create new architectures and discover interesting patterns.

The current system is already a useful research tool to test new ideas and architectures, although there is different aspects that we would like to improve:

- use rendering agents that utilise GPU instructions, thus enabling a fully featured GPU cluster using the existing distributing and performance agents.
- extend the reflective system to have an "intelligent user interface" that agents can modify, by partly specifying it in a pipeline.
- implement planning agents and in general more autonomous algorithms – add intelligent behavior to the distributing mechanism
- implement ontologies on top of the current system would be interesting to provide another information level agents could leverage
- work on collaboration scenarios, taking advantage of the current PDA client and Web client
- extend the current Volume Rendering pipeline to a more general graphic rendering pipeline (polygons)

Acknowledgements

Financial support for this work was provided by the UK Engineering and Physical Sciences Research Council through grant numbers GR/S46567/01, GR/S46574/01 and GR/S46581/01.

REFERENCES

[ADK+99] Martin Aeschlimann, Peter Dinda, Loukas Kallivokas, Julio López, Bruce Lowekamp, and David O'Hallaron. Preliminary report on the design of a framework for distributed visualization. *Parallel and Distributed Processing Techniques and Applications*, pages 1833–1839, 1999.

[App] Apple. Cocoa. `http://developer.apple.com`.

[ASA01] Mark Astley, Daniel C. Sturman, and Gul A. Agha. Customizable middleware for modular distributed software. *Communications of the ACM*, 44(5):99–107, 2001.

[ASF05] Apache Software Foundation ASF. Apache xml-rpc. `http://ws.apache.org/xmlrpc/`, 2001-2005.

[BBC+05] Ken Brodlie, John Brooke, Min Chen, David Chisnall, Ade Fewings, Chris Hughes, Nigel W. John, Mark W. Jones, Mark Riding, and Nicolas Roard. Visual supercomputing: Technologies, application and challenges. *Computer Graphics Forum*, 24(2):217–245, 2005.

[BDG+04] Ken Brodlie, David Duce, Julian Gallop, Musbah Sagar, Jeremy Walton, and Jason Wood. Visualization in grid computing environments. *IEEE Visualization*, pages 155–162, 2004.

[Cam95] Gordon Cameron. Modular visualization environments: Past, present, and future. *Computer Graphics*, 29(2):3–4, 1995.

[Dye90] Scott D. Dyer. Visualization: A dataflow toolkit for visualization. *IEEE Computerr Graphics and Applications*, 10(4):60–69, 1990.

[FK97] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[FK99] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2, "Computational Grids". Morgan-Kaufman, 1999.

[FKT01] Ian Foster, Carl Kesselman, and Steve Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[Fos05] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *proceedings of the IFIP International Conference on Network and Parallel Computing*, pages 2–13, 2005.

[FSF] FSF. GNUSTEP, a FSF implementation of the OPENSTEP APIs. `http://www.gnustep.org`.

[GAW05] Ian J. Grimstead, Nick J. Avis, and David W. Walker. Visualization across the pond: How a wireless pda can collaborate with million-polygon datasets via 9,000km of cable. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, pages 47–56, 2005.

[GFB05] Zahia Guessoum, Nora Faci, and Jean-Pierre Briot. Adaptive replication of large-scale multi-agent systems – towards a fault-tolerant multi-agent platform. In *Proceedings of the fourth international workshop on Software engineering for large-scalemulti-agent systems*, pages 1–6, 2005.

[HEvLRS03] Hans Hagen, Achim Ebert, Hendrik van Lengen Rolf, and Gerik Scheuermann. Scientific visualization – methods and applications –. In *Proceedings of the 19th spring conference on Computer Graphics*, pages 23–33, 2003.

[IBM] IBM. Autonomic deployment model. `http://www-306.ibm.com/autonomic/levels.shtml`.

[JJZ+04] Hu Jun, Gao Ji, Huang Zhongchao, Liao Beishui, Li Changyun, and Chen Jiujun. A new rational model of agent for autonomic computing. In *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 6, pages 5531–5536, 2004.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *IEEE Computer*, pages 41–50, 2003.

[KCBC02] Fabio Kon, Fabio Costa, Gordon Blair, and Roy H. Campbell. The case for reflective middleware. *Communications of the ACM*, 45(6):33–38, June 2002.

[KHRV04] Dieter Kranzlmüller, Paul Heinzlreiter, Herbert Rosmanith, and Jens Volkert. Grid-enabled visualization with gvk. *Across Grids 2003*, LNCS 2970:139–146, 2004.

[MF00] Isabel Harb Manssour and Carla Maria Dal Sasso Freitas. Collaborative visualization in medicine. In *Proceedings of The International Conference in Central Europe on Computer Graphics, Visualization And Interactive Digital Media (WSCG)*, pages 266–273, 2000.

[MGR+99] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, and Pattie Maes. Hive: Distributed agents for networking things. In *Proceedings of the First International Symposium on Agent Systems and Applications / Third International Symposium on Mobile Agents*, page 118, 1999.

[Mül03] Marcus Müller. XML/RPC framework for objective-c. `http://www.mulle-kybernetik.com/software/XMLRPC/`, 2002-2003.

[RKACM03] R. Rangel-Kuoppa, C. Aviles-Cruz, and D. Mould. Distributed 3d rendering system in a multi-agent platform. *Computer Science, 2003. Proceedings of the Fourth Mexican International Conference on*, pages 168–175, September 2003.

[RWB+05] Mark Riding, Jason D. Wood, Ken W. Brodlie, John M. Brooke, Min Chen, David Chisnall, Chris Hughes, Nigel W. John, Mark W. Jones, and Nicolas Roard. e-viz: Towards an integrated framework for high performance visualization. In *UK e-Science All Hands Meeting 2005*, pages 1026–1032, 2005.

[Squ] Squeak. `http://www.squeak.org`.

[SWB03] John Shalf and E. Wes Bethel. The grid and future visualization system architectures. *IEEE Computer Graphics and Applications*, 23(2):6–9, 2003.

[UFJK+89] Craig Upson, Thomas Faulhaber Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[Win99] Dave Winer. XML/RPC specification. `http://www.xmlrpc.com/spec`, June 1999.

[ZM05] Avelino Francisco Zorzo and Felipe Rech Meneguzzi. An agent model for fault-tolerant systems. In *Proceedings of the 2005 ACM symposium on Applied Computing*, pages 60–65, 2005.

Exposing Application Graphics to a Dynamic Heterogeneous Network

John Stavrakakis
jstavrakakis@vislab.usyd.edu.au

Zhen-Jock Lau
zhenjock@gmail.com

Nick Lowe
nickl@vislab.usyd.edu.au

Masahiro Takatsuka
masa@vislab.usyd.edu.au

ViSLAB, The School of IT
The University of Sydney

ABSTRACT

With the abundance of high performance personal computers, rendering thousands to millions of polygons per second is an inexpensive task. In recent years, there have been advances in networking technologies that have enabled applications to become distributed over a network and many applications require this functionality. These applications can range from driving a large display, collaborating over an internet, or supporting pervasive environments. Solutions currently exist in providing graphics over a network. However, they are usually targeted to satisfy particular problem domains or are otherwise difficult to adopt as applications require major adjustment. OpenGL ® is a graphics drawing library. Although many applications have made use of this API, few provide direct interaction within networked environments. In this paper we present Lumino, a framework that enables graphics from an existing OpenGL application to become available to a dynamic heterogeneous network. What differentiates Lumino from prior work is that it provides this functionality to existing unmodified applications at a very low level and is capable of supporting flow control, quality and scalability. Moreover, it is targeted at wide adoption and will be released under a Free Software license.

Keywords: Graphics, OpenGL, Network, Remote Rendering, Collaborative Environment, Visualisation, Grid

1 INTRODUCTION

Over the years computer graphics has seen many advances in the area of graphics hardware, software for new functionality in graphics APIs, and algorithms for new rendering techniques in complex modelling. An emerging trend for computer graphics is migration from single to multiple display devices or graphics accelerators. As this trend continues, a network will be needed to support this activity between multiple devices[19].

The OpenGL[17] API is an application programming interface that allows a programmer to access the features of the underlying graphics hardware. It has been accepted as an industry standard under which many applications have been developed. While the needs of local graphics resources have been satisfied, the advent of fast networking technologies has made it feasible to transport graphics over a network. As this became more apparent, applications have begun to utilise proprietary protocols for distributing graphics over a network. This does not extend to existing applications as they would require reimplementation and integration.

WSCG 2005 conference proceedings, ISBN 80-903100-7-9
WSCG'2005, January 31 – February 4, 2005
Plzen, Czech Republic.

Our solution Lumino, addresses the many issues involved with transporting OpenGL graphics over a network. It is able to provide fundamental services including flow control, state management to support dynamically connecting clients and peer-to-peer connectivity. It additionally enables applications making use of the OpenGL API to transparently distribute an OpenGL stream to multiple recipients who are able to dynamically connect and disconnect. As a result, applications need not be reimplemented, a scalable number of clients can share graphics of an application while quality graphics is provided. The parser, code generation, networking, and fundamental algorithms (for state management, encoding, decoding, and network transmission with flow control) are currently implemented in Lumino. Lumino will be made available under GPL[8], as complete OpenGL function coverage remains ongoing work.

The rest of this paper is structured as follows. We first assess existing solutions; outline our approach, present implementation details and their results. Finally, we conclude with a short discussion, and a brief outline of future work.

2 RELATED WORK

In this section we discuss previous work with particular emphasis on GLX [21] and Chromium [11]. These systems provide a foundation for some aspects of Lumino, but differ greatly in global design and application domain. This distinction is clarified in section 3. These systems have a number of features in common

with Lumino, but are not functionally equivalent. Both are freely available, use well-defined non-proprietary protocols for sending rendering command streams over a network, and are designed for applications that use OpenGL documented capabilities of existing proprietary systems.

2.1 GLX

A standard technique for remote display of 3D applications is GLX, the "OpenGL Extension to the X Window System". The X Window System[9] was developed to provide a network transparent user interface with rendering on a remote server. The X protocol is a client-server protocol in which applications are run on the client machine and display requests are sent to a server. This design allows application data and processing to be performed locally with window management and drawing to be handled transparently at the server. It is highly portable because any X client can connect to any X server, regardless of platform or operating system. Moreover, since all communication is handled by the client-side X library (Xlib) and the X server, applications do not have to be network aware.

GLX enables OpenGL applications to draw to windows provided by the X Window System. It is comprised of an API, an X protocol extension, and an X server extension. When using GLX for remote rendering, GL commands are encoded by the client-side API and sent to the X server within GLX packets. These commands are decoded by the X server and submitted to the OpenGL driver for rendering on graphics hardware at the server. Importantly, GLX provides a network transparent means of remote rendering to any X server that supports the extension. It also specifies a standard encoding for GL commands.

Figure 1 illustrates GLX. An application that uses the GLX API can send GL render requests to a remote X server that supports the GLX server extension. GL commands are encoded in the GLX packet, which is itself inserted into an X packet. Any number of clients can connect to an X server, but a client will only ever connect to a single X server. GLX is limited because of these characteristics. Rendering is always necessarily server-side and it cannot support GL command streaming to multiple remote displays.

2.2 Chromium

Chromium is a well established and widely used for rendering on display clusters. It is based on another technology called WireGL [10]. One of the major advantages of Chromium is that it enables users to construct a high-performance rendering system, which is capable of large scale complex rendering. Moreover, it can drive a large multi-display system to display high-resolution images. It is possible to create a remote rendering system based on Chromium and video streaming

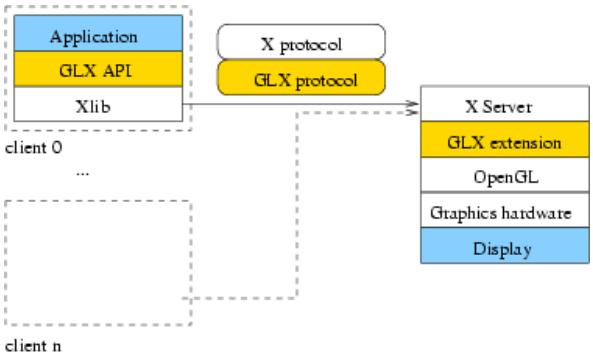


Figure 1: GLX is composed of a client-side API, a protocol for GL command stream encoding, and an X server extension (all components shown in orange). The application resides on the client machine and the display is connected to the server (both indicated in light blue).

or via its reflect SPU. However, this approach would be very inflexible and would not scale well in dynamically supporting multiple clients/rendering tasks.

Figure 2 illustrates Chromium's distribution model. It is a general node-based model for stream processing. A node accepts one or more OpenGL command streams (GL streams) as input and outputs GL streams to one or more other nodes. Each node contains one or more Stream Processing Units (SPUs) that modify the GL stream. Rendering or display can occur at any node in the graph. This depends entirely on whether the SPUs on the node perform rendering or display. One characteristic of Chromium that is not illustrated in the figure is that configuration of the graph is centralized and set at system initialization. This is suitable for dedicated clusters (with fixed, single-purpose resources), but not ideal for grid computing (with heterogeneous resources that are dynamically added and removed, and also available for other services).

Chromium follows the OpenGL Stream Codec (GLS)[6] to encode GL commands. GLS defines methods to encode and decode streams of 8-bit values that describe sequences of GL commands invoked by an application. Chromium, however, employs its own optimized protocol to pack all opcodes into a single byte. Hence, any stream produced becomes specific to Chromium. While this feature enables Chromium to achieve excellent cluster-based rendering performance, the portability of the generated stream becomes a limiting factor for extending Chromium into other OpenGL based projects.

2.3 Commercial remote visualization products

There are a number of commercially available solutions to provide visualization capabilities to remote users, such as SGIs OpenGL Vizserver[18], Teraburst[14] and

Figure 2: Chromium has a flexible configuration system that arranges nodes in a directed acyclic graph (DAG). Each node can take input from, and send output to, multiple nodes. A client node takes GL commands from an application and creates a GL command stream. A server node takes GL commands from another node (and usually renders the stream on local hardware). Because Chromium is usually used for remote display, this diagram shows rendering and display at a server node. However, it is important to note that rendering (and display) can occur at the any node.

Sun Microsystems' VisGrid[13]. However, they all require a proprietary protocol and a communication channel to exchange and control visual information. In contrast, most grid services are based on open source applications. Moreover, many resources are heterogeneous in their nature. Hence, the software infrastructure providing such grid services must allow users and research communities to construct and provide grid resources without being restricted to a particular platform technologies. As many commercial remote rendering solutions are used along with the data grid, computational grid and access grid, they are not a part of grid services. Therefore, users have to access such facilities in a conventional client/server manner and the system is normally configured at the beginning of each session.

Although these commercial and proprietary solutions provide a high and stable performance, many graphics and visualization projects continue to use commodity based approach, such as Chromium, due to the following reasons as described by Samanta et al.[16]:

- **Lower-cost** Price performance ratio of consumer hardware vs. custom architectures

- **Technology tracking** Rate of performance improvement in consumer hardware exceeds that of custom architectures, consumer hardware is also easier to upgrade

- **Modularity & flexibility** Networked systems allow PCs to be added or removed, and enables PCs to be used as other resources not only for rendering

- **Scalable capacity** The aggregation of compute power, storage, and bandwidth capacity in a network of PCs scales easier than that of multiprocessor architectures, which are limited by the complexity of their interconnects.

3 OUR WORK

Current widely used cluster rendering systems are targeted at the provision of a high-performance rendering capability to a local display system. Solutions to remote rendering currently exist. However, they are based on proprietary commercial solutions, and are difficult to integrate with open source grid infrastructures.

We have developed an open source remote distributed rendering system named Lumino. Figure 4 shows the basic system diagram of how Lumino interacts with a client application and remote rendering system. At any given instance, it appears very similar to the Chromium model. However, it is a dynamic system with a fundamentally different processing model also detailed in figure 3.

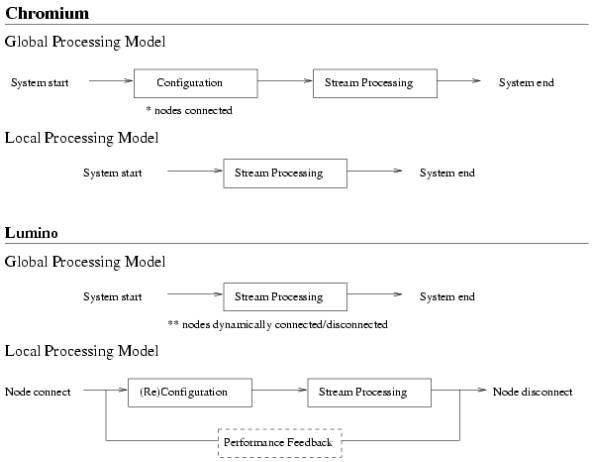


Figure 3: This diagram illustrates the differences between the Chromium and Lumino distributed processing models. Chromium adapts a global processing model composed of an initial configuration stage followed by stream processing on static nodes. Lumino uses a dynamic local reconfiguration model that allows nodes to connect and disconnect at runtime. This removes the need for an initial global configuration stage. Note that node connection and disconnection in this diagram refers to connection to the system (stream processing at each node can involve input and output to/from many other nodes).

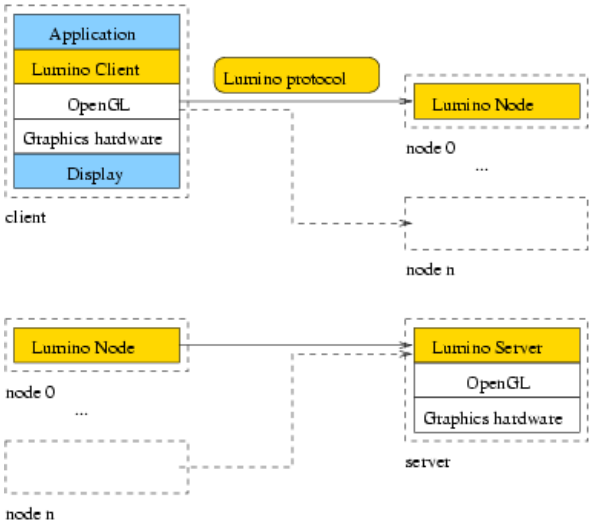


Figure 4: At any given time, a Lumino network configuration looks similar to a Chromium configuration. It has nodes arranged in a graph, in which each node can take input from, and send output to, multiple nodes. A client node takes GL commands from an application and creates a GL command stream. A server node accepts GL commands from another node. It is important to note that this diagram represents a system-wide snapshot and that node connections are dynamic. Also of note is that rendering (and display) can occur at any node.

3.1 Framework

The OpenGL stream is a sequence of serialised OpenGL commands intercepted by the application. Each command in the stream uses a unique identifier to recognise the OpenGL API call it represents, termed an *opcode*. The Lumino framework consists of several components; each strictly requires an interface to an OpenGL stream. For the OpenGL stream, we adopted the GLX protocol as it can readily interface with existing GLX servers. It is also well-defined for interoperability of future OpenGL functionality and its opcode identifiers. To describe processes among machines in the system, we term them producers or consumers of an OpenGL stream.

3.2 State Management

State management is a component that aims to maintain a representation of an applications OpenGL context, that is, the state of the OpenGL machine. This influences how rendering is done, for example which lights are enabled, what are their attributes etc.

State management is required to keep the state of separate OpenGL machines synchronised. This enables the rendering process for *each* machine to produce the same results that are intended by the application. In addition, state management allows any consumer to dynamically connect and render frames of the application.

The OpenGL API contains commands that influence the state of the OpenGL machine. These commands are captured from the stream and appended to an OpenGL state stream; representing the OpenGL context. The state stream must remain consistent, transparent and incur low computational cost. These are the requirements of state management. In being consistent, the state stream remains in order, thus allowing their execution in a new OpenGL context to result in the same graphics state. It is transparent such that only a GL stream is interfaced. State is arranged in an unknown manner, and when a consumer requires the entire state or a state update, a GL stream is returned. That is, no explicit state information is sent.

To simply append state commands to a data structure would lead to a memory explosion. Our backend implementation uses a data structure that can efficiently manipulate commands such that they are deleted when certain conditions are satisfied. For example, when GL commands operate on matrix transformations that are later reset with the identity matrix, the previous transformation is no longer required. There are many such relationships between OpenGL commands that cause redundancy in similar ways. Results indicate that the costs associated with performing checking and reduction can be implemented efficiently with the right data structure or at best with an entire emulation of the OpenGL state machine.

In providing state management, consumers are able to connect at any time. Our interface strictly remains a GL stream, thus consumers need only understand one protocol rather than deal with synchronisation from out-of-band communication. This idea extends the simplicity of designing a system where only graphics is outputted on the network interface. As many good OpenGL applications make heavy use of server side state; clients on low bandwidth connections are able to download the state environment as an OpenGL stream and then experience remote rendering without the impact of network bandwidth.

3.3 Flow Control

Bottlenecks evident between OpenGL applications and the rendering backend are typically dependent on data transfer rates on AGP/PCI buses, execution rate of the application, and deliverable performance of the graphics card. These bottlenecks also exist across networks, the only difference being that the network changes from local AGP/PCI to LAN/WAN. Bandwidth is a major problem for transferring graphics and it remains a critical goal to many existing solutions to minimise bandwidth usage. Prior approaches taken usually result in optimised proprietary protocols and limited extensibility for widespread use. Lumino supports an interoperable means to distribute graphics over a network. In coping with limitations of underlying network capac-

ities, flow control is available to compensate for low bandwidth consumers.

In heterogeneous environments, performance of different graphics hardware can vary greatly. By selecting OpenGL as the abstraction of graphics resources, we are able to use the API to gain performance feedback of the graphics hardware itself. The Lumino processing model permits each consumer to use out-of-band communication channel for relaying performance feedback information. This allows the consumer to control the bit-rate or frame-rate at which the rendering commands are received. While consumer oriented flow control is desirable, the producer may choose to throttle all consumers to a specific rate. This way, a producer remains in control for coordinating the resources of multiple consumers. This mechanism helps the producer, consumer, and the network in only utilising those resources required.

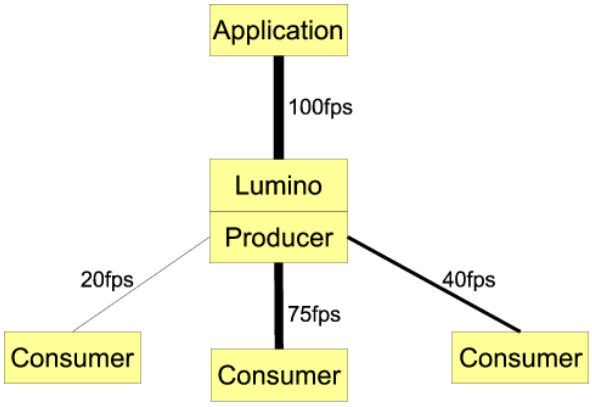


Figure 5: In this figure the application is capable of generating 100 frames per second. Lumino producer is able to negotiate rate control parameters among several consumers to satisfy any of bandwidth, processing or privilege requirements of all concerned.

3.4 Scalability

Scalability of network based graphics solutions can be defined as being resolution scalable, data scalable, consumer scalable or hardware scalable[7]. There are many limitations of existing solutions to provide remote rendering to a scalable number of consumers[20]. This is typically the case as they create centralised environments that put greater load on the server which would otherwise be uniformly distributed on the network. Our initial design sought TCP as a first solution in bringing graphics over a network. TCP offers many benefits to simplify implementation, but lacks point-to-point scalability.

Many P2P (peer-to-peer) solutions [1, 2, 3] allow consumers of a data stream, to become producers of that data stream to other parties. Lumino has applied the idea on sharing data stream, the only difference being that the stream is GL. This is achievable by being able to dynamically monitor and export the OpenGL context on-demand, enabling greater scalability as any consumer can now become a producer. The participation of new consumers is no longer burdening the single original producer; but rather, one of multiple producers as demonstrated in figure 6. By distributing the load among all parties involved, not only does scalability improve, but the requirements of the networks are eased which would otherwise be centralised and cause "hot-spots".

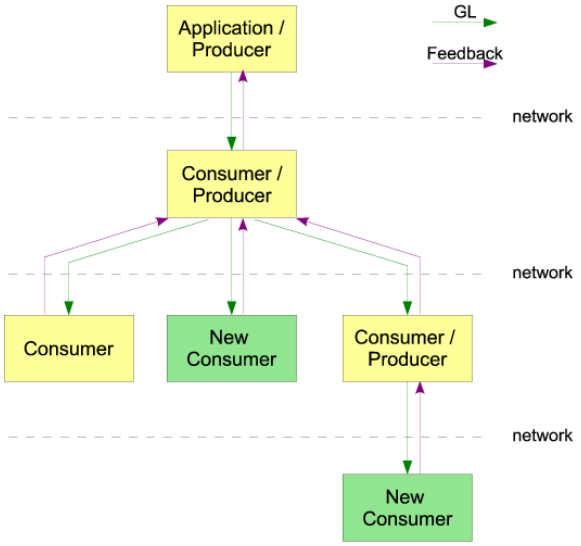


Figure 6: An application may be runnable on a device such as a laptop but is not capable of supporting more than one client due to processing limitations. To host this application for potentially hundreds of consumers, a more resourceful machine can be used to host the GL stream as a proxy to consumers. This alleviates any burden on the device to deliver processing and network resources. Furthermore, the GL stream can extend to other networks that can present the gateway to receiving the stream.

There remains a lingering issue of latency. For how long a chain of consumers will it be infeasible? This primarily depends on the underlying network hardware, but more importantly the amount of data generated by the application. The latency limitations are explored in the following section.

4 RESULTS

4.1 Processing overhead

Applications that run with Lumino are subject to overhead from encoding, state management and network I/O. We use three metrics to measure the performance of applications under Lumino, they are: frame rate, indicating how many frames per second can be rendered, execution rate, being the number of GL commands generated by the application, and byte rate, being the num-

ber of bytes generated after encoding. In this paper, we present experiments using two applications. The XScreenSaver[22] package contains various OpenGL programs that are used as screen savers. These are very simple with respect to demands and coverage of OpenGL. A more complicated and demanding application is the game application Tuxracer[4]. The machines used in testing are illustrated in table 1. Figures 7 and 8 describe the frame rate performance of Molecule and Tuxracer.

Machine	Specification
kat	P4 2.4GHz, 512MB RAM NVIDIA GeForce FX 5200/PCI/SSE2
duck	Celeron 2.2GHz, 256MB RAM ATI Radeon IGP 330
grudge	Dual P4 2.8GHz, 512MB RAM NVIDIA Quadro4 750 XGL/AGP/SSE2
calvin	P4 2.6GHz, 512MB RAM NVIDIA GeForce4 Ti 4600/AGP/SSE2

Table 1: Each machine has a minimum 100Mbps ethernet connection, and a Linux OpenGL driver with hardware acceleration.

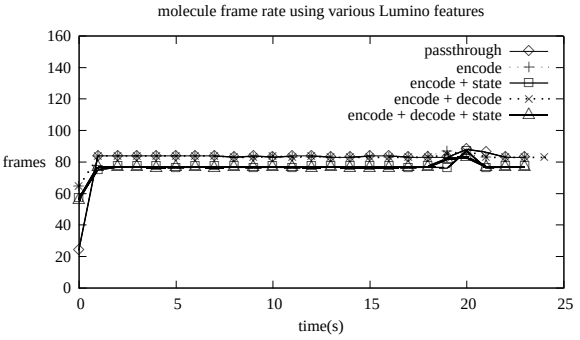


Figure 7: The Molecule application generates a very small amount of data, as a result the impact of encoding, state management and decoding is less noticeable when compared with its original performance.

4.2 Scalability Testing

While Lumino is able to support consumer scalability, the latency effects of intermittent processing between producers and consumers can affect display synchronisation. We devised an experiment that investigates the latency effects as the number of indirect consumers increases. In our efforts to demonstrate heterogeneous networking, we used two LAN environments: the School of IT (SIT), and ViSLAB shown in figure 10. In this experiment the networks and machines all differed other than running Linux (namely Ubuntu[5]) with 32-bit graphics drivers. We ran applications on a PC (kat) on the SIT LAN; the Lumino stream was then transmitted to a PC in ViSLAB (grudge), then to another ViSLAB PC (calvin), then to a laptop (duck) in

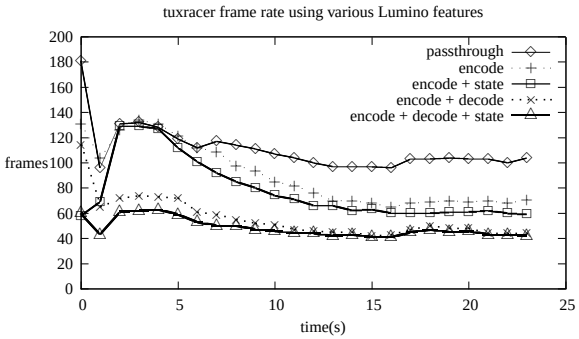


Figure 8: The Tuxracer application, is constantly submitting data to OpenGL. As a result the cost involved with encoding the OpenGL commands becomes a much larger issue than that of state management. It is important to note that a typical producer will be hosting the application by only performing encoding and state management. The lines with decoding indicate that decoding an already encoded OpenGL stream is marginally small. Consumers receive an OpenGL stream that is already encoded, thus the processing overhead is much smaller for them.

SIT, and finally to the original SIT PC kat. Figure 9 depicts this arrangement.

Figure 9: The Lumino stream begins at kat, it is forwarded to grudge, calvin, duck and finally again to kat. With the initial and resulting stream simultaneously rendered at kat we are able to observe the effects of latency over a 100Mbps network.

As the communication of Lumino remained TCP, the round-trip-times of each hop were measured with ping of maximum IPv4 sized packets shown in figure 2.

ping	rtt (ms)
kat->grudge	32.338
grudge->calvin	11.801
calvin->duck	31.399
duck->kat	35.989
total	111.527

Table 2: Average round-trip-time between machines detailed in table 1.

We modified gltext, to render a string representing the time in milliseconds. The round trip time difference for this application is measured by taking a snapshot of both renders. Shown in figure 11.

Our theoretical round trip time was measured at approximately 111ms. This did not explain the latency

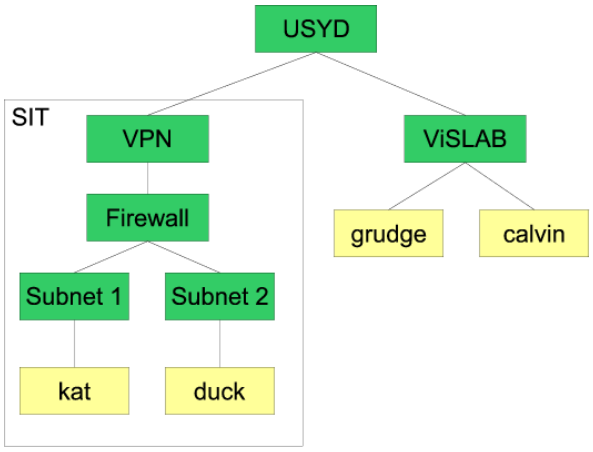


Figure 10: The organisation of the SIT and ViSLAB networks are bridged by a much larger USYD network, each uses different technologies including both Gbps and Mbps. We present these results knowing that the communication between all machines is limited to 100Mbps.

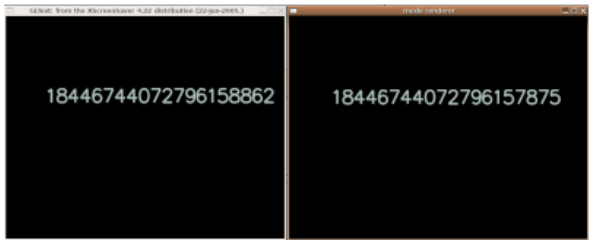


Figure 11: The left render is performed locally, whilst the right has been transmitted through 4 hops before rendering. The numbers from both indicate the current time from the source; the difference represents the latency which is approximately 987ms.

presented here. The factors that introduced latency include:

- Firewall processing - for the interests of SIT, packets are delayed at the firewall for security.
- TCP latency and processing - under TCP, processing of packets must continue to the application level, considering the burden on each machine there will be delay. In each machine the GL stream is received, the data is moved up through to the application level, and then pushed back down the stack for forwarding.
- Higher data throughput - the amount of data is proportional to the latency cost as a result of both transmission and additional processing.
- Incurred cost of state management has mostly affected the less powerful machines, namely the laptop duck. State management remains a backend modular component, as such other approaches are yet to be evaluated.

In exploring a more demanding application, we found Tuxracer provides a more network intensive latency test under the same conditions, shown in figure 12.

Figure 12: As before, the left indicates the local render, while the right has been transmitted through 4 hops before rendering. Using the rendered timer, the difference can be observed to be about 480ms; on average we found it to be approximately 940ms.

As Tuxracer incurs a similar latency cost with greater bandwidth requirement, we can conclude that Lumino is scalable, but remains bound to the underlying network technology.

To improve on the scalability, we are able to implement a multicast based replacement over TCP in a similar manner to Broadcast GL[12]. However, support for multicast networks are limited over an internet which would otherwise require reservation of IP addresses. In consideration of the user, this is not a requirement but an option.

5 CONCLUSION AND FUTURE WORK

We have presented Lumino, a framework that enables graphics from an existing OpenGL application to become available to a dynamic heterogeneous network.

Solutions available to providing graphics over a network are able to offer flexibility in their own domain. However, they exist with proprietary interfaces or have limited applicability in the broad resolution of network based graphics.

We have defined a unique and novel problem statement that Lumino addresses. Lumino is built from the aspects of existing solutions, the implementation of peer-to-peer scalability, flow control and dynamic connectivity are new features founded by Lumino.

Lumino already maintains large application support via the OpenGL API. It is complementary to existing technologies like that of Chromium and VNC. Moreover, it is established as a working product and model.

Its future development envisages innovative uses for transporting graphics over a network. As such, there are many research challenges that await Lumino, one of them describing a realtime system.

5.1 Future Work

There are many possible outcomes that Lumino is able to offer. In understanding that existing solutions pro-

vide focused objectives for application domains we are hoping to use Lumino to bridge such technologies together

- Transparent collaboration - applications are usually written without concern for network interoperability. As the software becomes a useful tool, VNC is used to view and control the application remotely.
- Lumino currently provides a one-to-many arrangement from single producer to multiple consumers. We are in the process of implementing a many-to-one and many-to-many arrangement, such that one consumer can have multiple producers for a single OpenGL context. We can interpret the scenario as "compositing" OpenGL streams. This permits geometry from one application to mix with geometry from another creating a hybrid rendered image. It presents an interesting gateway to a new breed of network enabled graphics applications.
- As the amount of geometry used in the render generates greater bandwidth, Lumino is not data scalable. The contrasting solutions are remote frame buffer such as VNC[15] where it is not resolution scalable. We aim to find a method by which we can gradually alter the bandwidth usage from sending geometry to sending images. This slider like control is desirable for consumers on heterogeneous networks.

REFERENCES

[1] BitTorrent. http://www.bittorrent.com/, 2005.

[2] eMule. http://emule.org/, 2005.

[3] The FreeNet Project. http://freenet.sourceforge.net/, 2005.

[4] Tux racer. http://tuxracer.sourceforge.net/, 2005.

[5] Ubuntu - linux for human beings. http://www.ubuntulinux.org/, 2005.

[6] Craig Dunwoody. The opengl® stream codec: A specification. Technical report, Silicon Graphics, Inc., October 1996.

[7] Matthew Eldridge, Homan Igehy, and Pat Hanrahan. Pomegranate: a fully scalable graphics architecture. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 443–454, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[8] Free Software Foundation. Gnu general public license. http://www.fsf.org/copyleft/gpl.html.

[9] X.Org Foundation. About the X Window System. http://www.x.org/X11.html, 2005.

[10] Greg Humphreys, Ian Buck, Matthew Eldridge, and Pat Hanrahan. Distributed rendering for scalable diaplays. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, page Article No. 30, Washington, DC, USA, 2000. IEEE Computer Society, IEEE Computer Society.

[11] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, July 2002.

[12] Tommi Ilmonen, Markku Reunanen, and Petteri Kontio. Broadcast gl: An alternative method for distributing opengl api calls to multiple rendering slaves. In *WSCG'2005: The Journal of WSCG*, volume 13, Plzen, Czech Republic, 2005. Science Press.

[13] Sun Microsystems. Sun fire^{TM} visual grid system architecture - building scalable and flexible high-end visualization systems. Technical white paper, Sun Microsystems, November 2003.

[14] TeraBurst Networks. Immersive visualization theater connectivity over the wide area network. http://www.teraburst.com/technology/wavs.pdf, October 2002. http://www.teraburst.com/technology/wavs.pdf.

[15] RealVNC Ltd. RealVNC: the original open-source cross-platform remote control solution. http://www.realvnc.com, 2005.

[16] Rudrajit Samanta, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 97–108, New York, NY, USA, 2000. ACM Press.

[17] Mark Segal, Kurt Akeley, Chris Frazier, Jon Leech, and Pat Brown. The opengl® graphics system: A specification. Technical report, Silicon Graphics, Inc., October 2004.

[18] Silicon Graphic, Inc. Opengl® vizserver 3.1: Appliation-transparent remote interactive visualization and collaboration. White paper, Silicon Graphic, Inc., Mountain View, CA, USA, April 2003.

[19] John Stavrakakis, Nick Lowe, Masahiro Takatsuka, and Zhen-Jock Lau. An On-demand Streaming Protocol for 3D Graphics on the Grid. APAC 05, 2005.

[20] John Stavrakakis, Masahiro Takatsuka, Nick Lowe, and Zhen-Jock Lau. Lumino: Platform independent remote distributed rendering framework. http://wiki.vislab.usyd.edu.au/moinwiki/Lumino, 2005.

[21] Paula Womack and Jon Leech. Opengl® graphics with the x window system®: Version 1.3. Technical report, Silicon Graphics, Inc., October 1998.

[22] Jamie Zawinski. XScreenSaver: A screen saver and locker for the X Window System. http://www.jwz.org/xscreensaver/, 2005.

Collision Avoidance and Surface Flow for Particle Systems Using Distance/Normal Grid

Tommi Ilmonen
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
Tommi.Ilmonen@hut.fi

Tapio Takala
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
tta@cs.hut.fi

Juha Laitinen
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
Juha.Laitinen@tml.hut.fi

ABSTRACT

Fire, explosions, and other special effects are often created with particle systems. In real-time applications the particle systems must be very fast to compute since otherwise the application cannot maintain reasonable frame rate. One part of this challenge is the collision detection between particles and the objects in the scene. We present a new approach to collision detection and surface flow effects for particle systems. In pre-processing phase we rasterize a 3D model into a distance/normal grid. The grid can be used for collision avoidance, to create surface drag and to simulate fluid flow around non-deforming objects. This method is not physically accurate, but it provides visually plausible results. The primary benefit of this method is that it is efficient and its performance is independent of the complexity of the model. This methods works well in real-time, in some cases surpassing the rendering speed of modern graphics hardware by order of a magnitude.

Keywords
Particle animation

1 INTRODUCTION

Particle systems are widely used in both off-line and interactive graphics to simulate fluid phenomena such as fire, explosions, smoke and clouds [Bur00a]. To be fully credible such effects need to interact with their environment.

To handle the collision of a particle with a surface one needs to use either 4D collision detection (taking into account the velocity of the particle) or surround the particle with some geometry (sphere or cube for example) that is used for collision detection. In both cases we must check collision of the particle against all surface primitives – a slow operation if the object is complex.

FULL Papers conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

If accurate collision detection is infeasible then one needs to turn to methods that trade accuracy for speed. The traditional approach to collision avoidance has been to identify intersecting objects and then apply collision resolution rules to achieve intersection-free state. Our approach to this problem is to create a combined distance/normal (DN) grid that is used to affect the particles as they fly close to the object. This grid can be used not only for collision avoidance, but also for other fluid effects such as drag and surface flow. many of the artifacts are not visible in the particle animations – the large number of particles and their overall fuzziness hides the imperfections of the physics simulations.

We developed this technique for an interactive particle-system installation where we needed high-performance collision detection. We were also interested in simulating fluid dynamics with the system, since many particle effects represent fluid systems. We concluded that any traditional collision detection system would be too slow for our purposes and a more efficient method was needed. We did not need a physically accurate system but one that is visually convincing. We could leverage the specific features of our installation since the scene was composed of rigid, non-deforming objects. These constraints led us to develop this method.

This paper is organized as follows: First we review existing collision detection systems. Then we introduce the distance/normal (DN) grid and the creation of such grids. After that we go through the effects that one can create with the distance/normal grid – collision avoidance, drag and surface flow. Finally we give some ideas for further development of the method.

2 BACKGROUND

Collision detection is one of the most fundamental problems in computer graphics and much research has been done to create optimal collision detection systems (for a comprehensive survey of techniques,see Hadap et al [Had04a]). While most of the systems handle various kinds of 3D objects there are are also collision detection systems that are specialized to particle systems.

Sims has described the general components of particle systems and also collision detection with spheres and planes [Sim90a]. Karabassi has built a system that performs exact 4D collision detection between particles and solid objects and collision avoidance between particles[Kar99a]. Like many others, her system uses spherical repulsive force fields to keep particles from hitting each other.

Distance fields are frequently used in robotics for collision avoidance. For example Greenspan[Gre96a] and Jung[Jun96a] have used voxel-based method for collision avoidance in robotics. These methods work by storing the distance of a surface from a voxel to the closest surface. In run-time the system checks how far the closest objects are and if there is a risk of collision the systems typically revert to ordinary intersection-based collision detection.

Steele has developed a system that does collision detection, avoidance and response with the aid of a vector field[Ste98a]. This method is similar in principle to our approach, but since Steele's system relies on a few simple geometrical field shapes it cannot be used with more complex objects.

Vector fields have been used since the seventies to visualize fluid flow. In these cases the flow field is first calculated with some physical modeling system and then quantisized into a voxel grid. While the data structures in this approach are almost identical to the distance/normal grid the way the grid is calculated, interpreted and used is different. In the classical vector-grid approach the grid alone determines the path of the particles while we use more complex rules to get collision avoidance and surface flow that are not predefined. Figures 6 and 7 demonstrate how we can create variable effects with one grid – a feature that the older grid-based methods do not support. With our approach it is possible to merge other forces (variable wind, explosions, gravity) with the grid.

3 DISTANCE/NORMAL GRID

The most common way to avoid collisions is to detect and resolve them as they happen. Another method is to use repulsive gradient fields that prevent collisions from happening in the first place. In past, force fields have been used to avoid collision between simple objects like spheres and it has not been possible to create force fields around objects of arbitrary shape. The novelty of our approach lies in the idea of using a distance/normal grid to represent objects of arbitrary shape and a collection of algorithms that can be used for collision avoidance, drag, and surface flow. Compared to previous approaches our system performs faster by taking advantage of both distance and normal information that is stored in the grid. This method works if one of two the colliding objects is a particle – if both were complex objects we would eventually come up with yet another spatial-division collision detection system for rigid bodies.

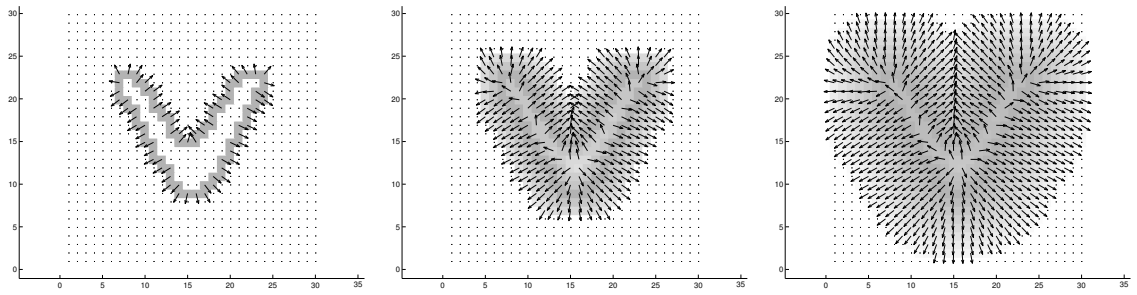
In the grid each voxel contains a three-dimensional unit-length vector (**N**) that indicates the normal of the object and a distance value (l) that tells how far is the closest surface from the center of the voxel. At run-time we can get the distance and normal in any point by looking up the voxel that contains the particle. The computational complexity of the method is O(1) – its run-time performance is not affected by the complexity or shape of the object.

On the general level we first rasterize the object to a 3D grid. Then we thicken the surfaces of the object to cover more voxels and create a DN-grid. The resulting grid is stored to a file. The grid is read from the file in run-time and used with the collision avoidance and surface flow algorithms to make the particle system react to its environment.

3.1 Grid Construction

A critical step in this method is the creation of the DN grid. There are many ways to surround an object with a voxel grid. The basic requirements of the grid are that the normal vectors close to the object should be perpendicular to the surface and the normal vectors should change smoothly outside the surface and follow its overall shape. The grid should be dense enough to capture all relevant details of the object.

We have used two methods to create such grids from polygonal models. The first method is based on first rasterizing the object into the grid and storing surface normal at each voxel. When more than one polygon



(a) Initial distance/normal grid.

(b) Distance/normal grid after one round of blurring (blur kernel with linear attenuation and radius of 3).

(c) Distance/normal grid after three rounds of blurring.

Figure 1: Distance/normal grid around an object (from the same model as in figure 3, with grid resolution of 30x30x30). Each arrow represents the direction of the normal vector **N** at that point and color indicates distance from the surface (darker is closer).

hits the voxel the surface normal is approximated as the average of all normals. We then perform 3D blur on the vectors in the whole grid (exactly analogous to blurring a picture with three color components). We have used small (radius from 2 to 4 voxels) convolution kernels with linear attenuation. The vector grid can be blurred multiple times to smear the direction of the vectors and to spread the field around the object. After each blur round the normals in the voxels that are exactly on the edge of the the objects are returned to their initial value. This step is done to guarantee that the normals are perpendicular to the surface near the object. After the normal vectors have been blurred we normalize them to unit length. At this stage we also set the distance values to indicate distance from the surface. If the voxel is inside the object, then we use negative distance value.

Figure 1 shows a slice of a DN grid that we have created with this method. The arrows show the direction of the field at each voxel and the darkness of the voxel indicates the distance from closest surface. As can be seen this method creates a smooth vector field that follows the shape of the object. By using high-resolution grids we can create DN fields that capture the details of the object but are smooth further away from the object.

This method works usually well but fails to work properly if the object is too thin. In such cases the grid must be extremely dense since the back and front faces of an object must be rasterized to different cells. If the object has near-zero thickness (for example piece of paper) then this method does not work at all – the DN field field will only work towards one direction.

This problem can be addressed with a different method to for calculating the DN grid. With the second method we first rasterize the object into a grid but in each voxel we only store a potential value to indicate that the voxel is occupied. Then we proceed to thicken the potential field by blurring the values (as in previous case). Finally we turn the potential field to a force grid by calculating the normal (gradient) at each voxel with difference method.

The motion of the objects needs to be taken into account when transforming the particle location and velocity from world coordinates to the local object coordinates. Each independently moving object must have a specific DN grid that moves with it.

The objects can also be rotated and translated freely without disturbing the shape of the grid. One can also scale the object uniformly, and the thickness of the DN field will be reduced accordingly. Objects cannot be sheared since shearing changes the angles between the normal vectors and the surface.

When retrieving the value of the DN field in the location of the particle, one can either use the value of the closest voxel to represent both the normal and distance to the closest surface or interpolate these values from the voxels surrounding the particle. We have not noticed any visible difference between these two methods and due to performance reasons we only use the nearest neighbor.

Even though the complexity and shape of a 3D object has no impact on the performance of the system there are objects that do not fit well into this scheme. Objects with very thin extrusions (for example a pencil) can be problematic since the fact that particles are bounced outside the object becomes more visible. A more difficult class of objects are the ones that have multiple thin extrusions that are close to each other (for example trees). In these cases the fields of neighboring extrusions will be merged and this may result

in a field that prevents particles from penetrating the volume of the object even though the object is sparse and the volume could be penetrated.

At run-time the grid is used to represent the object and appropriate rules are used to create bounces, drag and surface flow. The simulation is carried out in discrete time intervals and the rules are applied at each time step. If the grid is too thin then a particle may pass through it without being affected by the field. In practice this implies that the maximum velocity of the particles and the update interval of the physics engine needs to be known when the field is being calculated. While this is an obvious limitation it is not a severe problem since in most cases the creator of the 3D application knows the magnitude of velocities that the particles will have and can adjust the thickness of the field accordingly.

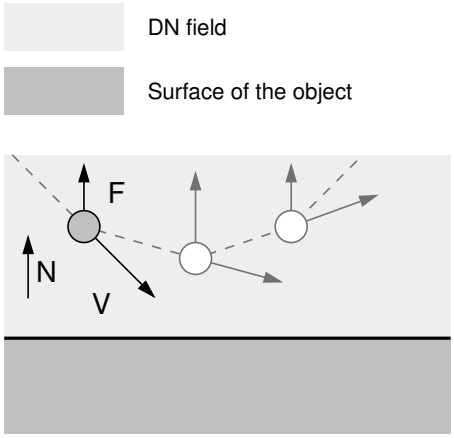


Figure 2: Particles are repelled away from the surface with the spring method. Vector **N** is the surface normal, **V** indicates how far the particle travels during one frame and **F** is the force that is applied to the particle.

3.2 Collision Grid Compression

The grids that we generate are often fairly large. This has a negative effect on performance since it increases the time to load a grid from a disk and easily fills the CPU cache memory. To counter this problem we developed a tree-based compression method.

This method is used as a post-process after the grid has been created. We analyze the grid to find areas where the normal vectors have either not been set or point to similar direction. In these areas the grid nodes are erased and instead a single vector is used to represent all vectors in that node. We also turn the distance/normal -information into a plane equation to avoid problems related to increased cell size. In practice the grid is dense only in areas where the object surface has high curvature.

We have used a two-level tree with tunable division parameters. Increasing the number of levels causes more indirection when accessing tree nodes, but may also reduce memory consumption.

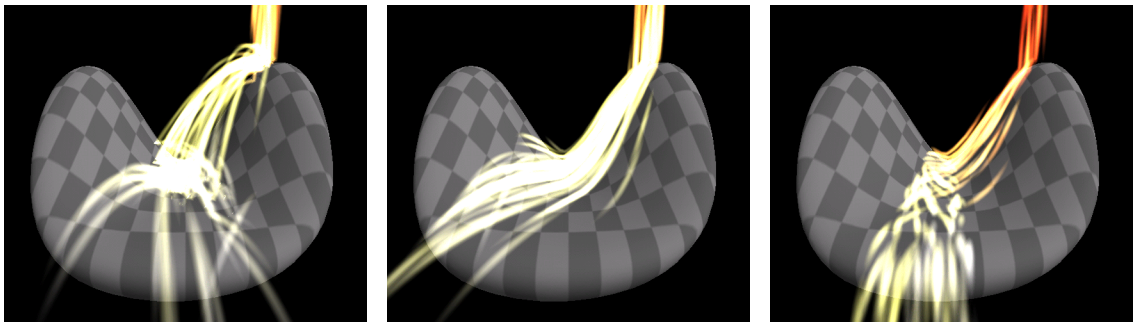
3.3 Collision Avoidance With Spring Method

There are two methods that are useful for avoiding collisions – the spring method and the impact method. The spring method is illustrated in Figure 2. This method works by using the force field as a spring – the deeper the particle passes into the field the more force is applied on it. As a result the particle is thrown back from the surface. The major drawback of this method is that the trajectory of the particle lacks the abrupt change that is caused by impact to a surface. Instead the path of the object is a paraboloid near the surface. Physically this can be understood as an elastic surface that yields as the particle hits it. Problematic artifacts are that a fast particle may fail to bounce from the surface or it may dive temporarily beneath the surface. The exit velocity of the particle is correct if the system is updated with very high processing rate. In practice this is seldom the case, but luckily minor variations in the bounce trajectories make the physics simulations more credible if anything.

3.4 Collision Avoidance With Impact Method

Another method to bounce the particles from a surface is with impact method. With this method we first check if a particle is inside the DN field. If it is, then we use the surface normal and apply direct bounce (or impact) to the particle. This method can be tuned to take into account the velocity of the particle: If the particle is flying with great velocity towards the surface it is bounced as soon as it enters the field. Thus slow particles are bounced close to the surface and fast particles closer or farther away depending on how they to move in the grid as illustrated in Figure 4. The velocity vector V is adjusted to become W after collision with the implied surface.

The impact method is generally superior to the spring method since it causes the physically valid abrupt change in particle velocity and slow particles are bounced close to the surface. In practice one seldom notices that the particles are reflected above or below the surface – the eye does not realize the inexact bounce point of the fast particles and the slow particles are reflected close to the surface as they should. Another factor that helps is that particles are typically rendered as textured billboards, lines or 3D polyhedra. Thus the particle should not bounce at exactly at



(a) Particles collide from a bouncy surface ($\beta = 0.7$).

(b) Particles collide with a non-bouncy slippery surface ($\beta = 0$).

(c) Particles collide with a non-bouncy surface with high drag ($\beta = 0$, $\delta = 2$, $\theta = 1$).

Figure 3: The effect of different bounce and drag properties (60x60x60 grid).

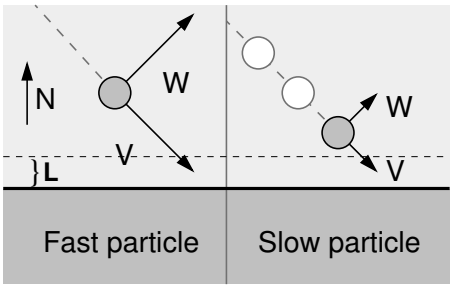


Figure 4: Particles are bounced with the impact method. Fast particles are bounced as soon as they enter the field and slower particles are reflected close to the surface. Vector **W** is the velocity after collision and L is collision threshold.

the surface, but rather above it to take into account the non-zero size of the particle.

3.5 Drag

When two solid objects (or an object and a fluid) interact they usually exert drag. It is easy to do drag calculations with the DN grid. If the particle is closer than a given threshold to the surface then we consider that it is in contact with the surface and drag force is applied on the particle. This method can be combined with the bounce methods since they do not disturb each other. Figure 3 shows combination of using the above impact method for collision resolution and surface drag.

3.6 Surface Flow

Often particles are used to represent fluid effects. In these cases one should use flow equations to determine fluid flow around the objects.

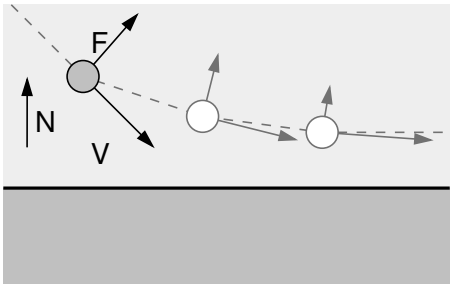
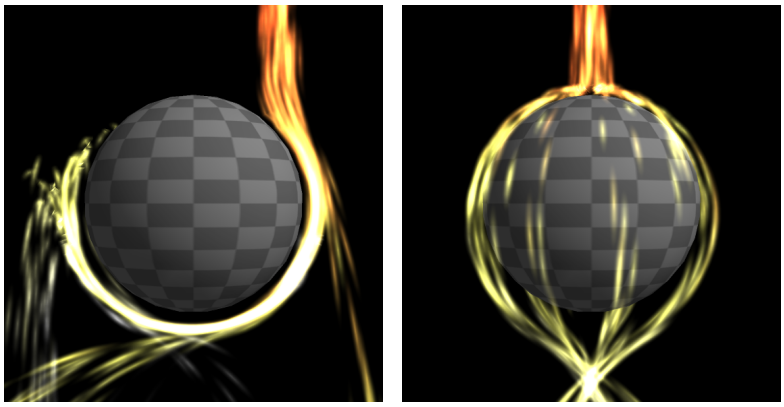


Figure 5: Distance/normal field is used to simulate surface flow by attract a particle to a tangential path.

The DN field is useful for implementing surface flow around objects. By surface flow we mean flow close to the object. In this case we apply a force that turns the particle to a trajectory that is tangential to the surface (Figure 5). The force vector **F** should be perpendicular to the velocity vector of the particle. Figures 6 and 7 show the effecs created with this approach.

We have found it is often good to have different coefficients for incoming and outgoing particles (ρ_{in} and ρ_{out}). These parameters can be varied to get different kinds of flows around the object. The following code example corresponds to figure 5 and is also used in figures 6 – 8.

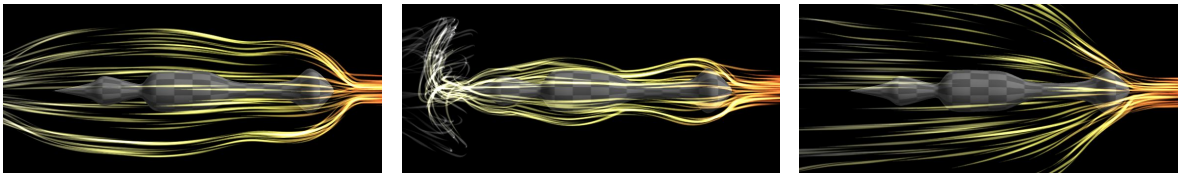
$\gamma = \mathbf{N} \cdot \mathbf{V}$
$\mathbf{E} = (\mathbf{V} \times \mathbf{N}) \times \mathbf{V}$
$\mathbf{E}_n = \mathbf{E} \ / \ |\mathbf{E}|$
`if` $(\gamma < 0)$
$\mathbf{F} = \rho_{in}\mathbf{E}_n$
`else`
$\mathbf{F} = \rho_{out}\mathbf{E}_n$
`fi`



(a) Particle stream coming to the side of the sphere.

(b) Particle stream falls on top of the sphere.

Figure 6: Particles flow around a sphere (45x45x45 grid). Identical grid is used in both cases.



(a) Strong incoming tangetial force (ρ_{in} = 12) and weak outgoing tangential force (ρ_{out} = 3).

(b) Weak incoming tangetial force (ρ_{in} = 3), strong outgoing tangential force (ρ_{out} = 6) and reduced flow distance. To the left of the shaft one sees turbulence that is caused by excessive tangential grab force.

(c) Weak incoming tangetial force (ρ_{in} = 3) and negative outgoing tangential force (ρ_{out} = -3).

Figure 7: Particles coming from right flow around a shaft (40x40x40 grid). Identical grid is used all cases.

Figure 8 shows the collision avoidance, and surface flow effects together. Collision avoidance is used to keep particles from penetrating the objects and the ground and surface flow is applied to make the particles flow around buildings in a perceptually credible way. This example shows how the DN field can mimic the results of physical systems with high credibility and great performance. In past such effects have been created by evaluating simplified versions of Navier-Stokes -equation which is by several orders of magnitude slower.

This method cannot be used to approximate general fluid-behavior since this method does not address important flow features such as turbulence, edge vortices or colliding fluid streams.

3.7 Performance

The operation of grid-based collision avoidance system is very fast. At run-time we only need to map the location of each particle to corresponding voxel in the grid, retrieve the normal- and distance values from the grid and apply the necessary rules. All of these are fast constant-time operations.

The primary issue for real-time applications is the memory footprint. A 50x50x50 grid that contains four 32-bit floating point numbers per cell takes 2 Mb of memory. A normal PC loads such a grid in 1-3 seconds from the hard disk. If the application must read grids at run-time from a file the few seconds needed to read one grid can be a problem. The grid size also exceeds the cache sizes in most CPUs, resulting in access to the slower main memory of the computer.

Figure 8 is the heaviest simulation in this paper with about 800 particles. Besides the DN field that represents the buildings it has a force generator that pushes particles away from the center of the explosion, gravity and air drag generators. In this system the graphics hardware sets the limits on the performance. An ordinary 1,5 GHz desktop PC can run the dynamics simu-

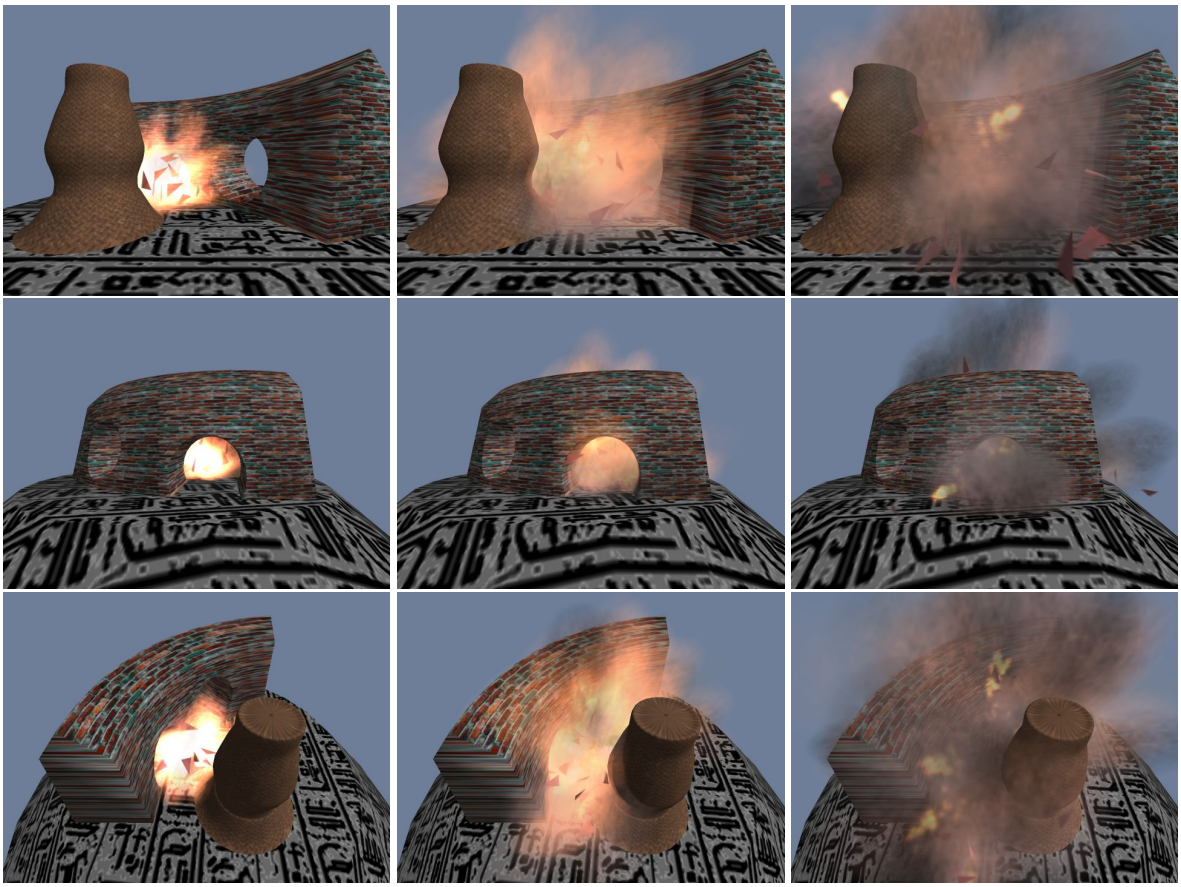


Figure 8: An explosion between objects – each row shows the explosion from one viewing angle at three times (60x60x60 grid, 800 particles). Note how the gas particles flow around the corners of the objects.

Collision System	FPS	FPS loss	Memory usage
None	1800	0	None
Compressed plane grid	1250	550	670kB
Normal grid	950	850	3.7MB
AABB collision detection	450	1350	Not known

Table 1: Performace of different systems in the scene of figure 9.

lation 800 times per second, but the graphics hardware (NVidia GeForce FX 5700) limits the frame rate to 50-80 Hz due to intensive fill-requirements (1024x768 window size, no anti-aliasing).

We have also tested the grid approach against classical collision-detection systems. In this benchmark we used a freely available AABB-based collision library "Opcode", that claims to be a high-performance tool for the task [Ter03a]. We used the ray-triangle intersection test in Opcode. The particle system was a minimal test system that contained simple particles (no color or texture changes) and simple dynamics (only gravity and one collision object). The system was ran without rendering to minimize the effect of rendering on the performance. A snapshot of the scene is in figure 9. The scene was designed to be a difficult case for the collision detection systems — all particles are close to the object and bounding-box -based early-out methods do not work. The test results are in table 1.

As expected the vector grid approach is clearly faster than the classical approach. Surprisingly the compressed vector grid was faster than the normal grid. This may be due to its more simple internal logic (it lacked drag and surface flow calculus) and/or better cache hit ratio.

The grid construction of a 50x50x50 takes a minute or two depending on the parameters (blur radius, unmber of iterations etc.). We have not experimented with optimization or benchamarking of this code since this work is always done in pre-processing phase.

3.8 Implementation

We have implemented the DN field with C++ to a particle engine that already had support for visualization, displaying 3D objects etc. It took 2600 lines of code to add all the features that have been described in this paper. Most of the code is needed to rasterize the 3D object into the grid and to calculate the grid – the collision, drag and surface flow implementations take about 100 lines of code.

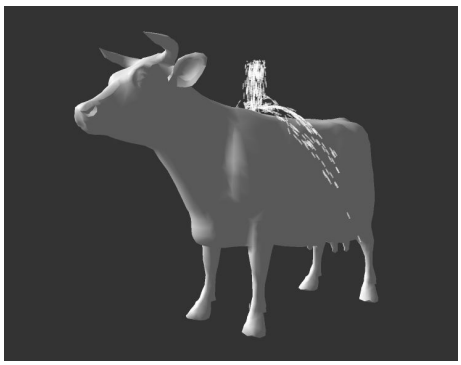


Figure 9: The test scene used in performance comparisons (12 000 triangles).

4 FURTHER DEVELOPMENT

In section 3.1 we presented two methods to generate the DN grid. One could come up with new methods by applying theories that simulate electric fields etc.

At the moment there is a trend to move computations from the CPU to the graphics hardware. The DN grid method might fit to this approach – the field can be stored as a 3D RGBA texture to the graphics hardware. The equations that govern the behavior of particles are very simple, implying that they can be implemented in the graphics hardware. While it is probable that graphics hardware could be used to run the collision detection the possible performance benefits remain to be seen.

5 CONCLUSIONS

The DN grid method can be used in a number of ways to create perceptually valid physics simulations for particle systems. This approach relies on and leverages the features of particle systems. We have shown that the DN field is useful for creating solid-body dynamics (i.e. bounces) and fluid-effects (surface flow and drag) with a single data structure. It is especially suited to situations where several particles are moving in a complex static scene and computational efficiency is an issue. It is not a 4D collision detection system but by using the range information we can create effects beyond pure 3D collision detection.

ACKNOWLEDGEMENTS

This work has been funded by the Academy of Finland.

References

[Bur00a] John van der Burg. Building an advanced particle system. *Game Developer*, March 2000.

[Had04a] Sunil Hadap, Dave Eberle, Pascal Volino, Ming C. Lin, Stephane Redon and Christer Ericson. Collision detection and proximity queries *GRAPH '04: Proceedings of the conference on SIGGRAPH 2004 course notes*, ACM Press, 2004.

[Gre96a] Michael Greenspan and Nestor Burtnyk. Obstacle count independent real-time collision avoidance. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, volume 2, pages 1073–1080. IEEE, 1996.

[Jun96a] Derek Jung and Kamal Gupta. Octree-based hierarchical distance maps for collision detection. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 454–459. IEEE, 1996.

[Kar99a] Karabassi Evaggelia-Aggeliki, Papaioannou Georgios, Theoharis Theoharis, and Alexander Boehm. Intersection test for collision detection in particle systems. *Journal of Graphics Tools*, 4(1):25–37, 1999.

[Sim90a] Karl Sims. Particle animation and rendering using data parallel computation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, pages 405–413. ACM Press, 1990.

[Ste98a] Kevin L. Steele and Parris K. Egbert. A unified framework for collision detection, avoidance, and response. In *WSCG'98 Conference Proceedings*, volume III, pages 517–524, 1998.

[Ter03a] Pierre Terdiman. Document named Opcode.html located in http://www.codercorner.com/Opcode.htm. Referenced 2.1.2006, Last updated 2003.

Efficient Occlusion Culling using Solid Occluders

Georgios Papaioannou Athanasios Gaitatzes Dimitrios Christopoulos

Foundation of the Hellenic World
Poulopoulou 38
11851, Athens, Greece

gepap@fhw.gr gaitat@fhw.gr christop@fhw.gr

ABSTRACT

Occlusion culling is a genre of algorithms for rapidly eliminating portions of three-dimensional geometry hidden behind other, visible objects prior to passing them to the rendering pipeline. In this paper, an extension to the popular shadow frustum culling algorithm is presented, which takes into account the fact that many planar occluders can be grouped into compound convex solids, which in turn can provide fewer and larger culling frusta and therefore more efficient elimination of hidden geometry. The proposed method combines planar and solid occluders using a unified selection approach and is ideal for dynamic environments, as it doesn't depend on pre-calculated visibility data. The solid occluders culling algorithm has been applied to commercially deployed virtual reality systems and test cases and results are provided from actual virtual reality shows.

Keywords

Hidden surface removal, visibility, virtual reality, games, dynamic environments.

1. INTRODUCTION

In dense or large three-dimensional environments, the visible geometry consists of many hundreds of thousands of polygons. Fortunately, most of these primitives are not visible simultaneously from an arbitrary vantage point. If the amount of actually invisible (hidden) polygons that are nevertheless sent to the graphics hardware for rendering is kept low, the performance of the application can gain a significant boost. The algorithms used for culling these surfaces fall into three categories, back face elimination, view frustum culling and occlusion culling, the later being not so trivial as the rest.

Occlusion Culling

In occlusion culling, geometry that is hidden behind objects closer to the camera point is discarded before being subject to depth sorting algorithms, although this geometry may pass the other two culling tests.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

Most occlusion culling methods use planar occluders [Hud97a] and their projections on the viewing plane. These planar occluders are often the polygons of the blocking geometry. Culling tests take place either in 3D space (viewer coordinate system, see Fig. 1) or in the resulting discrete image space, after projecting and rasterising the occluders and the potentially hidden geometry in an image buffer. The Hierarchical Z-Buffer [Gre93a] and the Hierarchical Occlusion Map [Zha98a] are two good examples of the later approach. Other techniques rely on the concept of space partitioning in cells, which are visible through openings, named portals, as in [Lue95a]. Binary Space Subdivision is often utilized to automatically segment space into cells based on the geometry [Tel91a]. The related visibility culling techniques may use pre-calculated information or clip entire cells on the fly. BSP and cell-and-portal techniques work very well for enclosed environments, such as building interiors and are thus favored by the game development community. Occlusion culling techniques may pose some restrictions on navigation freedom (non-arbitrary view location) in order to gain simplicity and performance, as is the case in [Dow01a]. A more extensive presentation of the different occlusion culling methods is clearly beyond the scope of this paper but the interested reader can find a thorough and comprehensive survey and comparison of occlusion culling algorithms in [Coh03a].

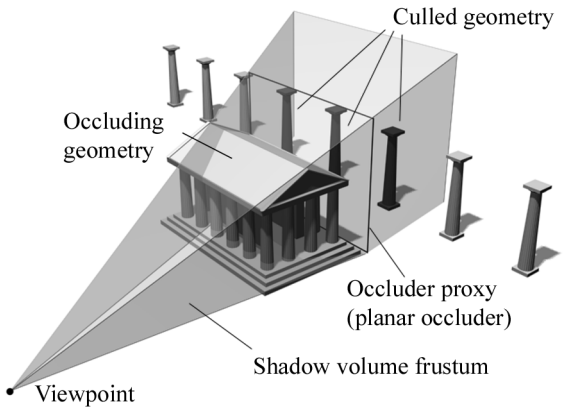


Figure 1. Principle of the shadow frustum culling method.

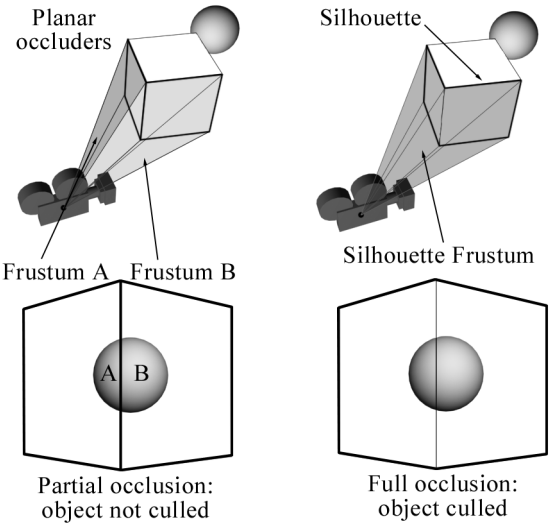


Figure 2. The generation of the culling frusta from convex solid silhouettes eliminates many cases of partial occlusion.

Occlusion tests using small, independent occluders, such as geometry polygons or view-dependent planar occluders, are inefficient because an object may be partially hidden by two or more occluders and therefore not discarded, but totally hidden by the joint surface that the occluders may produce (Fig. 2). To address this, fusion algorithms have been proposed by many authors, as in [Sch00a] and [Kol00a]. These methods try to combine the resulting frusta (in 3D space) or produce aggregate footprints (in image space) of the occluders in order to maximize the culling effectiveness of closely packed but otherwise unrelated occluders. Joining either the view-dependent occluders or the resulting frusta or projected polygons (image-space footprints) involves expensive operations that have to be performed at each frame. Furthermore, containment tests with non-convex (or multiple) frusta or non-convex polygons that are generated from the joint projected occluders are inherently more complex and time consuming when compared to simple individual convex occluder tests. Joining and visibility testing can be simplified by generating approximate convex aggregations of occluder clusters but in this case, an error is introduced which may become intolerable, especially when the viewer comes close to the occluders.

The method for occlusion culling presented here relies on the use of convex solid occluder shapes in a scene to block away geometry fragments (or hierarchies of objects) from arbitrary view points. It extends the work of Hudson et al [Hud97a] for occlusion culling using shadow frusta to address the inefficiency of partial visibility discussed above, while avoiding the expensive joining calculations. Shadow frustum culling uses – usually convex - planar occluder polygons, which can be part of the visible scene geometry or dummy (proxy) occluders, in order to create semi-infinite conical frusta with edges emanating from the viewpoint and passing through the polygon's vertices (Fig. 1). A geometrical entity, which can be anything from a simple polygon to a scene-graph hierarchy of complex objects, is blocked if its bounding volume is completely contained within the frustum. In the case of a hierarchy of objects, if it cannot be safely discarded as a whole due to partial containment into the shadow frustum, the tree is traversed in search of a potentially fully hidden sub-tree. Hudson et. al. [Hud97a] proposed a dynamic occluder selection mechanism which maintains a small list of most efficient occluders for each frame, selected among the full set of planar occluders using proximity, alignment and area coverage criteria. More about this selection mechanism and its extension to the solid occluder paradigm will be discussed in section 3.

Motivation

In many static or dynamic environments and especially in the case of outdoor scenes, where the geometry is often blocky (buildings, cars etc), clustered and possibly sparse, simple occlusion culling performs very poorly due to the partial occlusion phenomenon (Fig. 2): Two or more adjoining planar occluders may partially hide a distant object but the combined occlusion area of them may hide it completely. This object cannot be eliminated if the frustum for each occluder is created separately, and the joining of frusta is, as mentioned, an expensive operation. Pre-calculation methods solve this problem but cannot be effectively applied to dynamic environments.

2. METHOD OVERVIEW

To overcome this limitation, instead of individual planar polygons, convex solid occluders were used, such as boxes and cylinders as visibility blocking proxies for large isolated structures. For the moment, solid occluders are manually defined (modeled). A convex solid, when projected on an arbitrary plane, is guaranteed to produce a convex polygon. The convex frustum of the projected polygon is the union of the frusta that would be generated from the individual planes of the faces of the solid occluder, thus bypassing the need to merge frusta in order to avoid partial occlusion.

Solid occluders are easy and efficient to define in open space environments and therefore the method is most suitable for such scenes. In enclosed (indoor) static environments, although the application of the method is equally feasible, it does not provide any gain when compared with the simpler cell-and-portal occlusion culling schema. Shadow frustum culling using solid occluders is inherently compatible with moving geometry as the solid blockers can be transformed in space in the same manner as the rest of the geometry, without any computational overhead.

For each frame, the solid occluder frustum is generated as follows (Fig. 3). The view-dependent silhouette of the solid occluder is extracted by determining the edges belonging to adjacent polygons, which are not both visible or hidden simultaneously:

$$edge(tr_i, tr_j) \in Silhouette \Leftrightarrow \left[\vec{N}_i \cdot (\overline{P}_{i0} - \overline{C})\right] \cdot \left[\vec{N}_j \cdot (\overline{P}_{j0} - \overline{C})\right] \leq 0 \quad (1)$$

where $\vec{N}_k$ is the normal vector of triangle tr_k, $\overline{P}_{k0}$ the triangle's first point and $\overline{C}$ the viewpoint.

The silhouette edges do not lie on the same plane in general. Therefore, a cap (near plane) for the semi-infinite frustum must be approximately constructed, based on the relative position of the viewer and the silhouette points. If the near plane is too close, geometry that is contained in the convex hull of the edge poly-line may be falsely eliminated. Similarly, choosing a plane near the average of the silhouette points can be a disastrous selection for elongated solid occluders in the view direction for the same reason (Fig. 4). Therefore, the near plane of the frustum is chosen so that it passes through the furthest point of the silhouette line relative to the viewer. The normal vector of the near frustum cap is the average directional vector between each silhouette point and the viewpoint $\overline{C}$:

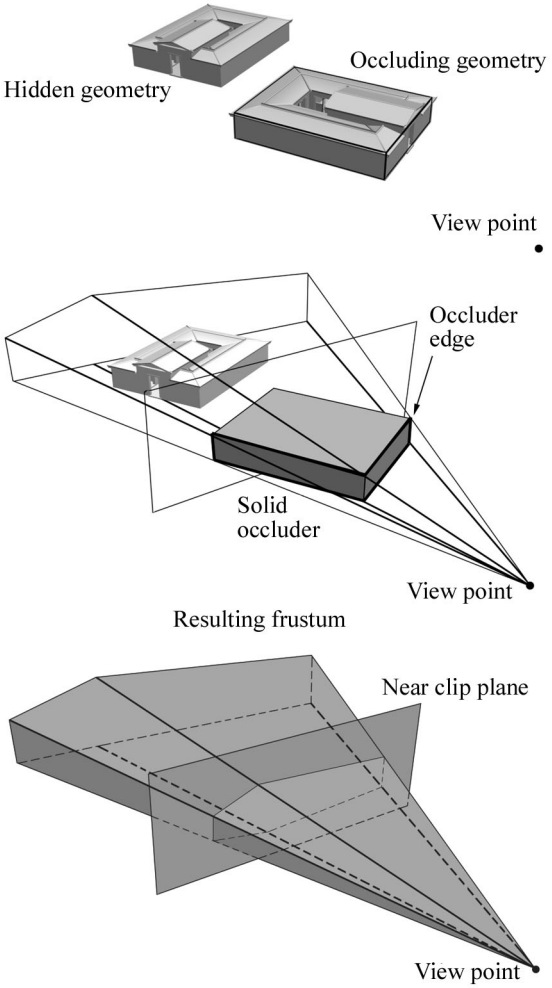


Figure 3. Frustum creation, from silhouette determination to (convex) frustum generation. The near cap of the frustum passes from the furthest silhouette point to the camera position.

$$\vec{N}_{near} = \sum_{i=0}^{K-1} \left(\overline{C} - \overline{S}_i\right) \Big/ \left\| \sum_{i=0}^{K-1} \left(\overline{C} - \overline{S}_i\right) \right\| \quad (2)$$

where $\overline{S}_i$ is the *i*-th point out of the *K* silhouette vertices.

The effectiveness of the solid occluders becomes apparent when the viewpoint is moving among buildings or other isolated obstacles at inspection distance (near). Planar occluders would mostly produce partial occlusion when the viewer is not facing the main sides of the geometry straight on. Most of the time, we view structures and moving objects from odd angles and this is where the solid occluders technique provides a unified and contiguous frustum to take into account all planar sides at once.

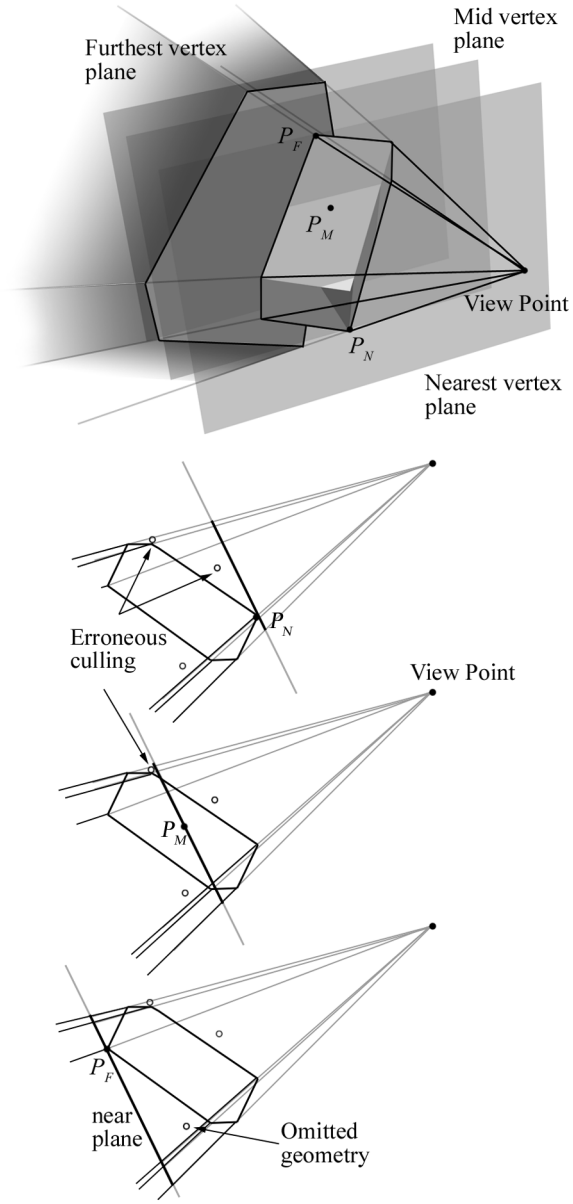


Figure 4. Near frustum cap selection.

3. OCCLUDER SELECTION

As explained by Hudson et al. [Hud97a], a scene may contain too many occluders for a real-time application to be able to test each object against each one of them. For instance, in a finished virtual reality production, such as the "Walk through Ancient Olympia" by the Foundation of the Hellenic World (FHW) [Gai04a], a virtual world may contain more than 200 occluder planes and solids. Therefore, an optimal set of occluders has to be selected for each frame at run-time in order to keep the number of "active" occluder primitives to a minimum. For this task, a ranking function f_{planar} and f_{solid} has to be devised for each type of blocking primitives that takes into account the solid angle of the frustum. For planar occluders, Hudson et al. use the area-angle approximation presented by Coorg and Teller [Coo97a]:

$$f_{planar} = \frac{-A\vec{N}\cdot\vec{V}}{\left\|\vec{V}\right\|^2} \qquad (3)$$

where A is the area of a planar occluder, $\vec{N}$ is its normal vector and $\vec{V}$ is the vector from the viewpoint to the center of the occluder. The criterion of eq. (3) provides a reliable measure of occluder importance. For solid occluders we use an approximation formula, which depends on the projection on the view plane of the solid occluder's volume *Vol* and the squared distance of the occluder from the viewpoint:

$$f_{solid} = \frac{Vol}{\left\|\vec{V}\right\|}\cdot\left\|\vec{V}\right\|^{-2} = \frac{Vol}{\left\|\vec{V}\right\|^3} \qquad (4)$$

Note that the above ranking function for the solid occluders does not depend on angular attributes as the near plane of the constructed frustum always faces the center of projection (see eq. (2)). f_{planar} and f_{solid} are balanced and do not need further biasing to become compatible. At each frame cycle, an active set of occluders, consisting of both solid and planar ones, is determined by evaluating and sorting the result of the corresponding ranking function for each one of them. Please note that due to time coherence and the fact that the occluder selection is not critical for the visual quality, it is not necessary to perform this sorting task at every frame. A typical size for the active occluders set is 10 – 20 occluders, depending on the field of view. For instance, our framework, running on a four-wall surround-screen platform, uses a set of 15 active occluders.

4. CASE STUDIES AND RESULTS

Various tests were run to compare the performance of the solid occluder algorithm with the one of the planar occluder scheme. As both planar and solid occluders are supported and incorporated in our application software that is used to drive the virtual reality shows at FHW, we were able to isolate the two cases under exactly the same application execution scenarios. The integration of the occlusion techniques within a complex scene-graph platform (SGI OpenGL Performer-based) also allowed for realistic performance testing, taking advantage of all the other optimizations available, such as frustum culling and location-sensitive geometry switching. The testing and benchmarking themselves took place in a four-

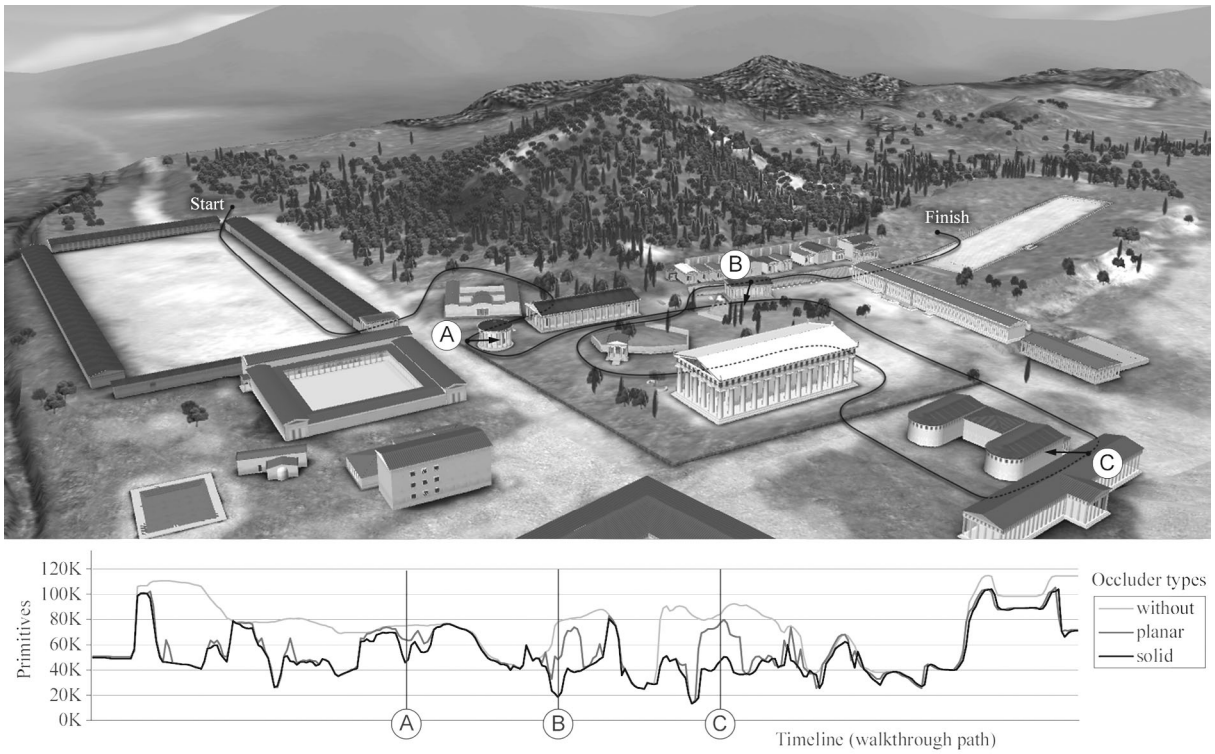


Figure 5. Sparse environment test case – Ancient Olympia. Solid occluders offer a significant culling performance improvement when the camera is moving near structures or when inspecting objects.

wall CAVE-like surround screen virtual environment. This means that view-frustum culling alone was not a very effective acceleration technique as the largest portion of the world was visible at any time in at least two displays. Therefore, this setup presented an ideal testbed for occlusion culling methods.

The test cases presented here are two extreme ones, one sparse rural environment - Ancient Olympia, as presented in the "Walk through Ancient Olympia" thematic show of FHW [Gai04a] – and a dense cityscape. Ancient Olympia consists of approximately $250 \cdot 10^3$ triangles, while the cityscape case uses about $500 \cdot 10^3$ triangles. The respective graphs in Fig. 5 and 6 show the number of primitives actually rendered (not culled) per walkthrough instance. For the statistic data capture, pre-recorded viewpoint position and orientation paths were used in order to maintain consistency among the experiments with different culling techniques.

In sparse environments, when the view position is far from the blocking geometry, the gain of the solid occluders over the planar ones is not significant. But when the viewpoint moves closer to objects, the solid occluders create larger, contiguous frusta, which effectively cull the hidden objects. Fig. 5 demonstrates this fact. At the view locations marked as A, B and C, the camera is facing towards the adjacent buildings. As planar occluders fail to cover both visible sides of a building simultaneously, hidden geometry is partially occluded, while in the case of the solid occluders it is fully occluded. Solid occluders are very convenient for structural geometry, like buildings, where convex polyhedra can be very intuitively placed or generated to tightly match the structure's shape.

In the second case study presented here (Fig. 6), a camera navigates through a model city, viewing the scene from a wide range of angles and locations. In order to keep the number of occluders low, each building is associated with a single though sub-optimal occluder. Using three box occluders instead of one, would have resulted in a more tight fitting of the geometry and therefore better culling results. The scene complexity of modeled environment is very balanced with regard to the view-direction due to the uniform distribution of three-dimensional models in space. As can be verified by the measured visible primitives at each path location, in this dense environment, there is an almost constant performance ratio between the solid and the planar occlusion culling method.

Apart from the computational cost of the geometry-frustum containment test, which is common to both planar and solid occluder methods, the overhead

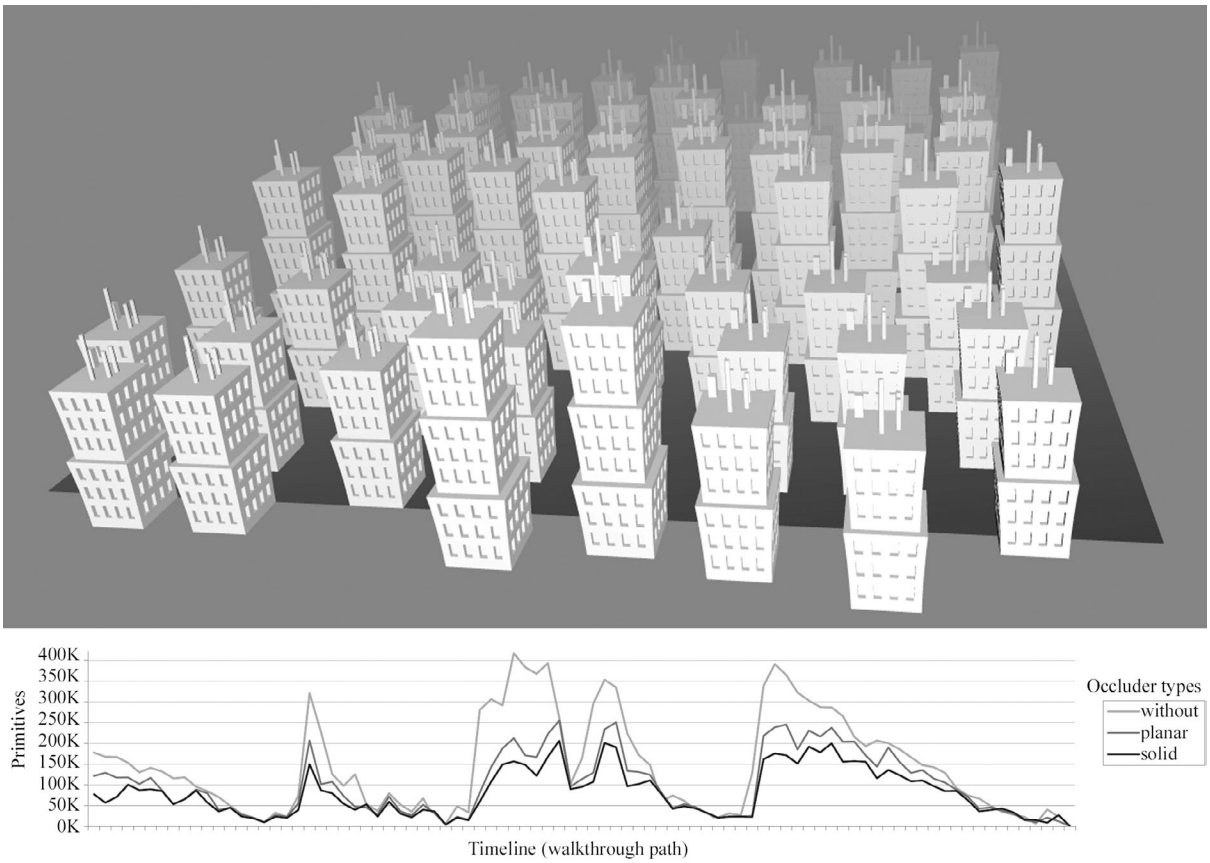


Figure 6. Dense environment test case. The chart shows primitives actually rendered (no occlusion / view frustum culling). There is an almost constant gain of the solid occluders over the planar ones.

introduced by the solid occluders comes from the determination of the visible edges of a convex solid, at each frame (see statistics in Table 1 and 2). In practice, this overhead is negligible because the silhouette is computed for very few and simple solid objects. This is ensured by the selection mechanism discussed in section 3.

The solid occluders culling method is an approximate algorithm, whose accuracy, compared to exact visibility calculation methods, depends on the selection of the occluders. In general, as occluders are enclosed in the shell of the actual geometric entities, the method is conservative, i.e. visibility is overestimated.

5. CONCLUSION

Even though the speed of today's graphics cards is increasing at a phenomenal rate, it will always be the case that researchers and game developers alike would like to push more geometry through the graphics pipeline than the card can handle at acceptable frame rates. Removing non-visible geometry before it reaches the pipeline is an

Occlusion Method	No Occl. Culling	Planar Occluders	Solid Occluders
Average primitives/frame	73096.9	57150.1	52882.0
Average Draw Time (ms)	100.57	89.71	85.14
Average Overhead (ms)	-	0.99	1.89

Table 1. Ancient Olympia Statistics

Occlusion Method	No Occl. Culling	Planar Occluders	Solid Occluders
Average primitives/frame	145488.8	100038.2	81064.2
Average Draw Time (ms)	136.80	92.60	78.26
Average Overhead (ms)	-	1.19	2.63

Table 2. Cityscape Statistics

important aspect of real time graphics. Occlusion culling in general and more specifically the shadow frustum culling method does exactly that, removing large portions of the scene-graph from the pipeline at a significant speed gain as we saw from the plots. Furthermore, the use of convex solid occluders pushes the rendering speed higher and provides an intuitive extension to the shadow frustum culling method.

One aspect that has not been investigated in this paper is the generation of frustums from non-convex solids. The silhouette determination of concave solids and the intersection of arbitrary geometry with a non-convex frustum is CPU intensive and well beyond the practical limits of real-time applications.

REFERENCES

[Coh03a] Cohen-Or, D., Chrysanthou, Y., Silva, C. T., and Durand, F. A Survey of Visibility for Walkthrough Applications, IEEE Trans. Visualization and Computer Graphics, 9, pp. 412-431, 2003.

[Coo97a] Coorg, S., and Teller, S. Real-time Occlusion Culling for Models with Large Occluders, in ACM Symposium on Interactive 3D Graphics proc., pp. 83-90, 1997.

[Dow01a] Downs, L., Möller., T., and Séquin, C. H. Occlusion Horizons for Driving through Urban Scenery, in ACM Symposium on Interactive 3D Graphics proc., pp. 121-124, 2001.

[Gai04a] Gaitatzes, A., Christopoulos D., and Papaioannou, G. The Ancient Olympic Games: Being Part of the Experience, in Eurographics 5th International Symposium on Virtual Reality, Archaeology and Intelligent Cultyral Heritage (VAST 2004) proc., pp. 19-28, 2004.

[Gre93a] Greene, N., Kass, M., and Miller, G. Hierarchical Z-Buffer Visibility, in ACM SIGGRAPH ’93 conf. proc., pp. 231–240, 1993.

[Hud97a] Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., and Zhang, H. Accelerated Occlusion Culling Using Shadow Frustra, in 13th Annual ACM Symposium on Computational Geometry proc., pp. 1–10, 1997.

[Kol00a] Koltun, V., Chrysanthou, Y., and Cohen-Or, D. Virtual occluders: An Efficient Intermediate PVS Representation, in Rendering Techniques 2000: 11th Eurographics Workshop on Rendering proc., pp. 59–70, 2000.

[Lue95a] Luebke, D., and Georges, C. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets, in ACM Symposium on Interactive 3D Graphics proc., pp. 105–106, 1995.

[Sch00a] Schaufler, G., Dorsey, J., Decoret, X., and Sillion, F. Conservative Volumetric Visibility with Occluder Fusion, in ACM SIGGRAPH 2000 conf. proc., pp. 229–238, 2000.

[Tel91a] Teller, D., and Sequin, C. H. Visibility Preprocessing for Interactive Walkthroughs, in ACM SIGGRAPH '91 conf. proc., pp. 61-69, 1991.

[Zha98a] Zhang, H. Effective Occlusion Culling for the Interactive Display of Arbitrary Models, Ph.D. Dissertation, Department of Computer Science, UNC-Chapel Hill, 1998.

Out of Core continuous LoD-Hierarchies for Large Triangle Meshes

Hermann Birkholz
Research Assistant
Albert-Einstein-Str. 21
Germany, 18059, Rostock
hb01@informatik.uni-rostock.de

ABSTRACT

In this paper, algorithms for the simplification and reconstruction of large triangle meshes are described. The simplification process creates an edge-collapse hierarchy in external memory, which is used for online reconstruction. The hierarchy indices are renamed after simplification, in order to allow fast reconstructions and the hierarchy is extended with information for view-dependent rendering.

The simplification makes no restrictions with the production of the hierarchy, but produces the same hierarchy as In-Core algorithms. The amount of memory, which is used for the simplification is adjustable.

Keywords

Out of Core, Level of Detail, triangle meshes, view dependent rendering

1. INTRODUCTION

Large polygonal meshes can easily be acquired with current 3d scanning hardware [Lev00]. Those meshes exceed the internal memory and the rendering capabilities of modern personal computers. For interactive visualization only partitions of the mesh can be used. In order to offer this, view-dependent approximations of the mesh must be fast computable. Such as for In-Core-meshes, continuous Level of Detail (cLoD) methods can be used to create fast view-dependent approximations. Because the mesh data does not fit into main memory these techniques must be adopted for such large meshes. Once created, a cLoD-hierarchy in external memory can be used for view-dependent online approximation of the original mesh. Therefor only visible parts of the hierarchy are read from external into internal memory and refined, until a time- or memory-limit is reached. These parts can then be visualized with the graphic hardware. In this paper a new simplification algorithm is presented, which creates cLoD-hierarchies for large meshes. This algorithm produces the same simplification hierarchy such as In-Core methods but it stores only partitions of the mesh in internal memory. The maximum memory footprint of the mesh is adjustable by the user. Furthermore it is demonstrated how to use the resulting hierarchy file to generate view-dependent approximations of the mesh.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

2. PREVIOUS WORK

In-Core cLoD Systems often build their hierarchies by collapsing edges [Hop96] or contracting vertices [Gar97] of the mesh surface. For each collapse/contraction operation an error value is computed, which determines the sequence of the collapses or of the contractions. For each edge collapse operation in manifold meshes two triangles are removed from the mesh surface. The two merged vertices and the removed triangles are stored in the LoD-hierarchy together with the error value.

For the simplification of large meshes various techniques have been presented. The spanned mesh simplification algorithm [San00] uses an external indexed mesh and an external heap with all collapsible edges, in order to determine the simplification sequence. The algorithm reads the first k (depending on internal memory size) edges from the queue and the mesh parts, which belong to the edges. Afterwards all

possible simplification operations for the edges, which are in memory, can be accomplished. Due to the nearly uniform distribution of the edges with small collapse errors, there will be hardly advantages from the locality of the read mesh parts. This will result in frequent mesh updates and load/store operations and thus long computation times. An advantage however is the fact that the simplification sequence is identical to In-Core algorithms.

In order to overcome the problem of the uniform distribution, hierarchical clustering is used to partition the mesh into locally connected blocks. Hoppe [Hop98] creates a block hierarchy and simplifies the mesh portions in the leaf blocks (edge collapse) except the edges, which cross the block borders. After the simplification, the leaf blocks are hierarchically merged and then simplified again. This is repeated, until the whole mesh is stored in the root cluster. Due to the forbidden collapses of edges, which cross the cluster-borders, this algorithm cannot limit the internal memory, used for the clusters. Furthermore the simplification sequence is limited by the cluster borders and might deliver bad results.

Cignoni [Cig02] suggests an external memory management based on octree subdivision. The management is able to overcome the problems with block borders and has a wide field of applications. For their simplification example however, they avoid the use of a heap structure for the correct simplification sequence, in order to take advantage of locality.

Another approach based on cluster hierarchies was presented by Lindstrom [Lin03]. His method consists of three steps. First, the mesh is clustered with an memory insensitive clustering technique, which is based on the clustering schema of Rossignac and Borrel [Ros93]. This step produces an uniform grid with cells, which contain either one or no vertex. In the second step, an octree is constructed over the grid, where the position of the merged vertices is determined with the QEM [Gar97]. In the third phase the hierarchy is used for view-dependent rendering of the mesh. The drawbacks of this method are the initial clustering, which might remove mesh details, if the grid-size is too high, and the octree schema, which produces hierarchies of a lower quality as edge collapse hierarchies.

So called “Processing Sequences” for computations on large meshes were introduced by Isenburg [Ise03]. Their mesh representation allows them to stream the mesh through internal memory and to apply changes to this local region. They show the simplification of large meshes as an example, but they neither produce a hierarchy, which can be used for reconstruction, nor they use a simplification order similar to In-Core methods.

The algorithm, which is presented in this publication produces simplification sequences, which are identical to In-Core algorithms ,and it takes advantage of local surface regions.

3. OUT OF CORE LOD Simplification

For mesh simplification, the half-edge collapse [Kob98] method is used. This means that an edge is collapsed to one of its vertices.

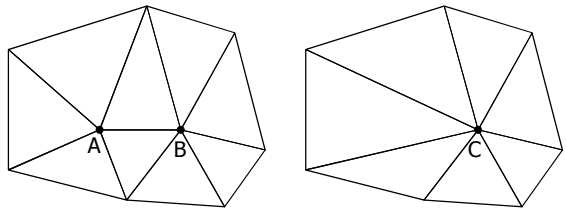


Figure 1. Half.-edge collapse

Figure 1 shows a half-edge collapse operation. The edge between vertex A and B is collapsed to vertex C. Vertex C is placed on the same position as vertex B. The simplification error of each vertex is computed with the Quadric Error Metrics [Gar97]. For each vertex all adjacent vertices are tested as potential collapse targets. The target which causes the smallest error value is stored for each vertex. Other error metrics, are applicable too. A simplification sequence equal to In-Core methods can be reached by executing only collapses which are locally minimal. That means, all adjacent vertices will cause higher or equal collapse errors. By iteratively applying locally minimal collapses, the hierarchy will become the same as if the collapses are applied in a global sequence (lowest first).

This fact can be used while simplifying large meshes. Local connected portions of the mesh can be read and all collapses for locally minimal errors can be applied.

Figure 2 shows the distribution of collapse errors for the “Stanford Bunny” mesh. The red/orange colors indicate high/medium collapse errors, while green colors indicate low collapse errors. As noticeable, the collapse errors are evenly distributed over the mesh. This also leads to an even distribution the locally minimal collapse errors (bright green dots). Furthermore the figure shows regions of low collapse errors that are surrounded by regions of low collapse errors. The surrounding vertices with high error values will not be collapsed until their neighbor vertices reach similar values. Higher collapse errors can only be reached by applying local simplifications in the low error regions.

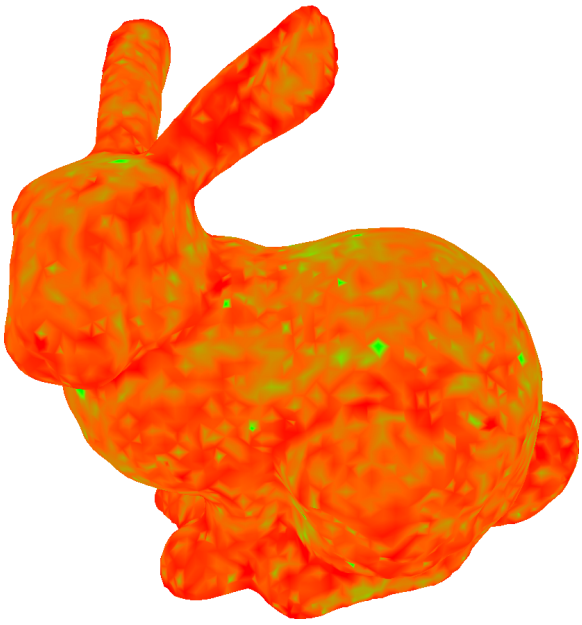


Figure 2. Collapse error distribution for the "Stanford Bunny" mesh

Starting with a small partition of the mesh in internal memory, all locally minimal collapses in the partition are executed or the partition is expanded when no locally minimal collapse error can be found. If the internal memory limit is reached, the memory data is written back to external memory and the process is restarted around the vertex with the smallest collapse error in the last partition.

In this implementation, **inner-vertices**, **border-vertices** and **near-border-vertices** are distinguished. For all vertices in internal memory all surrounding triangles are also read into the internal memory. For **border-vertices** not all vertices in the neighborhood have already been read from external memory. For this kind of vertex no collapse weight is computed, because not all possible collapse targets remain in memory. The **near-border-vertices** are completely surrounded by already loaded vertices but they have at least one **border-vertex** in their neighborhood. Because of the complete neighborhood, a collapse-target and –weight can be computed, but due to the at least one **border-vertex** in the neighborhood, it is impossible to determine if the weight is locally minimal. The **inner-vertices** have only other **inner-** or **near-border-vertices** around itself. Therefore locally minimum collapse errors can only be found among **inner-vertices**. Figure 3 shows a configuration with one **inner-vertex** (black), the surrounding **near-border-vertices** (black outline) and the **border-vertices** (stippled outline).

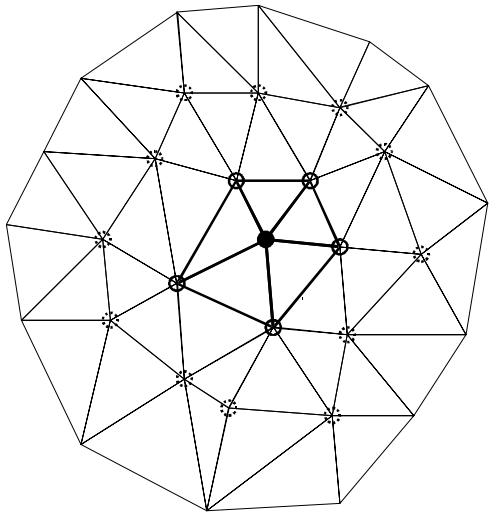


Figure 3. Local mesh partition with one inner vertex

Whenever no locally minimal collapse error can be found, the border of the local partition has to be expanded. Therefore the **near-border-vertex** with the smallest collapse error is chosen and all of its linked **border-vertices** are updated to **near-border-vertices**, by reading their neighborhood from external memory. This changes the state of the prior **near-border-vertex** to an **inner-vertex**, which can be checked for a locally minimal collapse error. Figure 4 shows the expansion of the upper right **near-border vertex** from figure 3.

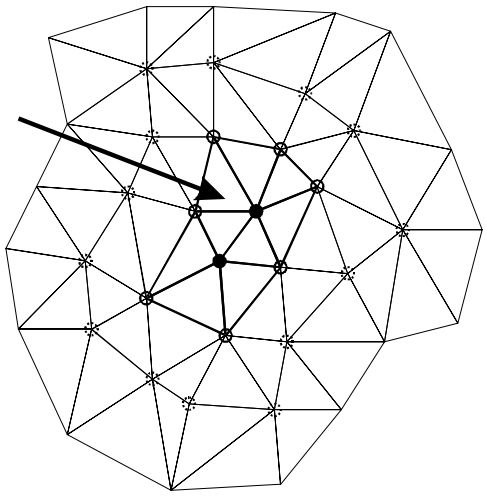


Figure 4. Local mesh partition with two inner vertices

Because always the **near-border-vertex** with the smallest collapse error is chosen, the In-Core part will always grow towards vertices with locally mini-

mal collapse errors. If a maximum number of vertices or triangles is exceeded in main memory, the In-Core vertices and triangles are written back to external memory. The simplification process then continues with the prior **near-border-vertex**, which would cause the smallest collapse error.

The original data is presented as an array of triangles and an array of vertices in external memory. For each vertex we also store the list of adjacent triangles. Changed and new vertices are stored separately and provided with hierarchy informations.

For each collapse operation the pair of merged vertices is stored in the resulting vertex. Furthermore the collapsed triangles are referenced in the resulting vertex. The collapsed vertices and triangles are removed from internal memory and all vertices, whose neighborhood changed (including the new vertex), are updated and checked for locally minimal collapse errors.

The simplification algorithm can be summarized as follows:

1. Choose and read a start vertex.
2. Read the adjacent vertices of the start vertex. (Make the start vertex to **near-border-vertex**.)
3. Expand the local partition by choosing the **near-border-vertex** with the smallest collapse error and update all **border-vertices** around it to **near-border-vertices**.
4. Execute all possible collapses among **inner-vertices**.
5. If the internal memory limit is reached, write internal data to external memory. Choose the **near-border-vertex** with the smallest error as new start vertex, read it and continue with step 2.
6. If further simplification is required, continue with step 3.

In the first step the vertex with the index 0 is normally chosen as start vertex. The second step reads the local neighborhood of the start vertex (triangles and vertices). After the second step, the start vertex is in the **near-neighbor-vertex** group and the other vertices are in the **border-vertices** group. The third step always expands the partition in internal memory, to find **inner-vertices**, whose collapse error is locally minimal. The fourth step executes all possible collapses within the **inner-vertices**. After that, a memory check is performed, to limit the use of internal memory. If no further local minimal collapse errors could be found, the algorithm restarts at the best position to find a local minimal collapse error. The last steps are repeated until a stop condition is reached. This can be for instance a maximum simplification error or a minimum number of triangles.

The indexed vertices and triangles of the internal partition are stored in AVL-Trees for fast access. With these trees, fast search operations can be performed in the local partition of the indexed mesh. The **near-border-vertices** are referenced in additional priority queues, for fast access to the vertex that will cause the smallest collapse error.

After the simplification, the remaining vertices and triangles are written to the hierarchy file for the use as approximation root.

Vertex Index Translation

In order to use the hierarchy file for the approximation of the original mesh, it has to be reconstructed top-down. This demands fast decisions how to distribute the triangles after each vertex split. This step can be accelerated by translating the hierarchy indices to an inorder structure. That means, the left subtree always contains only vertex indices which are smaller than, and the right subtree contains only vertices, which are greater than the according root node. Together with the information of the triangle vertices in the finest level, one can decide the correct child vertex in each split by only comparing vertex indices. All vertex indices in the left subtree are applied to the left child and the same is true for the right subtree.

View Dependent Rendering

After the translation of the hierarchy indices, some information for view-dependent rendering is added to the hierarchy. Fast view-frustum culling for each vertex in the hierarchy is supported with bounding-spheres. As soon as the bounding sphere of a vertex is outside the frustum, its complete subtree can be culled. For backface-culling and contour-based approximations, normal-cones [Xia96] for each vertex are computed. Whenever the cone points away from the viewer in the whole bounding-volume, the associated subtree can be culled.

The rendering process starts with the root of the hierarchy. All vertices, which were not collapsible and the triangles, which were not collapsed in the simplification process, construct the base mesh. These base vertices are put into a priority queue sorted according to their collapse error, which is divided by the distance to the viewer plane (greatest weight first). Vertices, which does not pass the view frustum test or the normal cone test are not put into the queue. Now the first vertex from the queue can be split into its child vertices iteratively and be replaced in the queue by its child vertices. The two associated triangles are appended to the triangle list during each split. Before the split, it is examined whether the child vertices are already in internal memory. If not, they are loaded from external memory. For internal memory management, an individual frame number is assigned to

all internal vertices. This variable is always set to the number of the frame, in which its associated vertex was last split. All parent vertices of a pair of leaf-nodes in the internal vertex tree remain in a priority queue, sorted by their frame number (smallest first). Whenever memory must be reallocated, the first vertex from the queue is used to remove its child vertices from memory. After that, its parent vertex is put into the queue, if both of its child-vertices are leaf-nodes in internal memory. This memory management allows to restrict the memory, which is used by the internal vertex tree.

The whole approximation process is terminated when a given time period elapses or a desired number of triangles is reached. So a minimum frame rate can be guaranteed. In order to reach higher frame rates it would also be possible to make use of frame-to-frame coherency. Therefor one must use the approximated vertex tree from the last frame and apply both collapse- and split-operations to it. Furthermore two priority queues are required. One for the split-candidates (greatest error first) and one for the collapse-candidates (smallest error first). Both queues must be balanced in each frame depending on the view parameters.

4. RESULTS

Simplification

A prototype implementation of the simplification algorithm was tested with several meshes. The results for small meshes were determined from the “Armadillo” mesh from "Stanford University Computer Graphics Laboratory". For medium sized meshes we used the “Asian Dragon” from “XYZ RGB Inc”. For tests with large meshes the “David” mesh from the “Digital Michelangelo Project” and a randomly created rough planet surface were used. Table 1 shows the data of the meshes.

Name	Vertices	Triangle
Armadillo	172,974	345,944
Asian Dragon	3,609,455	7,218,906
Rough Planet	67,108,866	134,217,724
David[1]	69,881,083	139,749,343

Table 1. Test meshes for simplification

The amount of triangles in internal memory was limited to 1,500,000 triangles for the local surface portions. This is equal to a memory consumption of around 230 MB. Due to the repeated tests for local minimal collapse errors, this algorithm executes of course much slower than In-Core algorithms. Table 2 shows the minimum, maximum and average "collapses per second", the overall simplification time and the hierarchy size in external memory for the test models. All test have been done on a Athlon 3800+ PC with 4GB of internal memory.

Name	Armadillo	Asian Dragon	Rough Planet	David
Avrg col/s	1663	1137	672	667
Min col/s	1397	797	97	103
Max col/s	1904	1825	1783	1764
Time h:m	0:02	0:53	27:44	29:14
Disc size in MB	19.7	413	7680	7855

Table 2. Simplification test results

The watertight meshes “Armadillo”, “Asian Dragon” and “Rough Planet” were all simplified to a tetrahedron as the base mesh. The base mesh of the “David” model consists of 1463 vertices after simplification due to small errors in the mesh surface. Small meshes, which can be cached completely by the operating system, show significantly higher collapse rates as large meshes. But the average rate of large meshes does not fall below 40% of the average rate of small meshes. Compared with other methods the achieved collapse rates are relatively low. But an implementation similar to the Spanned Mesh [San00] algorithm, which uses the correct collapse sequence, delivered much lower ratios, especially for large meshes (e.g 40 hours for the “Asian Dragon”). The main advantages of the new algorithm are, the hierarchy structure, which is equal to In-Core simplification algorithms, and the use of locality on the mesh surface for simplification.

View Dependent Rendering

The extraction performance for the external collapse hierarchy is comparable to In-Core variants. The only difference is, that parts of the hierarchy which does not remain in internal memory, have to be read during some frames from external memory. As soon as the desired hierarchy data remains in internal memory, there is no difference to InCore algorithms. Due to the fine granularity of the hierarchy, the extraction wont influence the desired frame rate very much.

All hierarchies were approximated with a maximum of 15,000 triangles. The relatively low number of triangles was chosen due to the approximation algorithm that does not make use of frame to frame coherency. An improved approximation algorithm should easily reach higher numbers of triangles at high frame-rates. Figures 5 shows approximated views of the four test models. The approximation detail can change while the view is moving, because

[1] Version of the mesh repaired with PolyMender [Ju04] written by Tao Ju

access to external memory may decrease the number of triangles, which are visible, in the desired frame time temporarily.

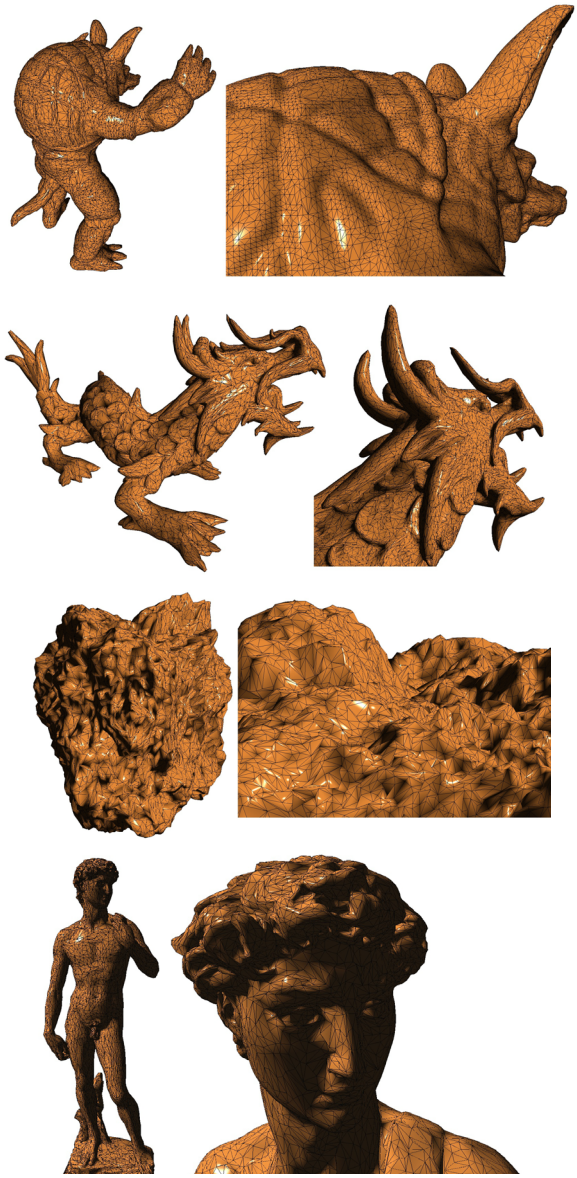


Figure 5. Approximated views of the test meshes (full view and cutout)

5. CONCLUSION

In this paper a new algorithm for the simplification of large meshes was described. It was shown how to use locally minimal collapse errors on the mesh surface in order to create collapse hierarchies, which are equal to ones produced by In-Core methods, while making use of locality on the mesh surface. The algorithm shows good simplification ratios compared with the algorithm of El-Sana [San00], which uses collapse sequences equal to In-Core algorithms. Furthermore it was described how to manipulate the hierarchy for fast triangle updates in the reconstruction process and its usage the hierarchy for view-depended rendering.

6. ACKNOWLEDGMENTS

I would like to thank Mr. Marc Levoy and the people working on the Digital Michelangelo Project for providing their models.

7. REFERENCES

[Cig02] P. Cignoni , C. Montani, C. Rocchini, R. Scopino, “External memory management and simplification of huge meshes”. IEEE Transactions on Visualization and Computer Graphics. 2002

[Gar97] M. Garland, P.S. Heckbert, “Surface Simplification Using Quadric Error Metrics”, SIGGRAPH ’97 Conf. Proc., pp. 209-216, 1997

[Hop96] H. Hoppe, “Progressive meshes“, Computer Graphics, 30(Annual Conference Series),pp. 99-108, 1996

[Hop98] H. Hoppe, “Smooth View-Dependent Level-of-Detail Control and its Aplications to Terrain Rendering,” Proc. IEEE Visualization ’98 Conf., pp. 35-42, 1998

[Ise03] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink, “Large Mesh Simplification using Processing Sequences”, IEEE Visualization 2003, pp. 465-472, 2003

[Ju04] T. Ju, “Robust Repair of Polygonal Models”, Proceedings of ACM SIGGRAPH, pp. 888-895, 2004

[Kob98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. „Interactive multi-resolution modeling on arbitrary meshes”, In Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 105-114, 1998

[Lev00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, “The Digital Michelangelo Project: 3D Scanning of Large Statues,” SIGGRAPH 2000, Computer Graphics Proc., pp. 131-144, 2000

[Ros93] J. Rossignac and P. Borrel, “Multi-Resolution 3D Approximation for Rendering Complex Scenes,” Geometric Modeling in Computer Graphics, pp. 455-465, 1993

[Lind03] P. Lindstrom, “Out-of-core construction and visualization of multiresolution surfaces”, Proceedings of the 2003 symposium on Interactive 3D graphics, pp. 93-102, 2003

[San00] J. El-Sana and Y.-J. Chiang, “External Memory View-Dependent Simplification,” Computer Graphics Forum, vol. 19, no. 3, pp. 139-150, 2000

[Xia96] J. Xia and A.Varshney, “Dynamic View-dependent Simplification for Polygonal Models”, Proceedings of IEEE Visualization, pp. 327-334, 1996

Multiresolution Wavelet Based Model for Large Irregular Volume Data Sets

Silvia Castro
Dpto. de Cs e Ing Comp.
Univ. Nac. del Sur
Avda. Alem 1253
Argentina
(8000) Bahía Blanca
smc@cs.uns.edu.ar

Liliana Castro
Dpto. de Matemática
Univ. Nac. del Sur
Avda. Alem 1253
Argentina
(8000) Bahía Blanca
lcastro@uns.edu.ar

Liliana Boscardín
Dpto. de Matemática
Univ. Nac. del Sur
Avda. Alem 1253
Argentina
(8000) Bahía Blanca
lboscar@uns.edu.ar

Armando De Giusti
Fac. de Informática
Univ. Nac. De La Plata
Avda. Alem 1253
Argentina
(1900) La Plata
degiusti@lidi.unlp.edu.ar

ABSTRACT

In this paper we propose a wavelet based model for large irregular volume data sets by exploiting a multiresolution model based on semi-regular tetrahedral meshes. In order to generate the multiresolution representation we use a wavelet based approach that allows compression and progressive transmission. Beginning with a semi-regular tetrahedral mesh $\Gamma_\infty(T)$ and applying the wavelet transform, we obtain a representation that consists of a coarse base mesh $\Gamma_0(T)$ and a sequence of detail coefficients obtained from the sucessive decomposition of the mesh at different levels of resolution. The base mesh is the one at the lowest resolution and it does not have the connectivity subdivision property. The wavelet decomposition is obtained by defining a wavelet basis over tetrahedra generated by a regular subdivision method applied to an initial tetrahedron T. The obtained basis is a Haar-like one and forms an unconditional basis for $L^p(T,\Sigma,\mu)$, $1<p<\infty$, being μ the Lebesgue measure and Σ the σ-algebra generated from the tetrahedron T by the chosen subdivision method.

Keywords

Volume Modeling, Multiresolution Modeling with Wavelets, Irregular Volume Modeling, Wavelets

1. INTRODUCTION

Three dimensional scenes contain highly detailed geometric models for many applications such as Internet 3D models for complex virtual environments, collaborative CAD, interactive visualization, etc. This situation motivates the development of 3D surface and volume models in order to meet requirements like effective use of disk space and network bandwidth, as well as substantial reduction of network transfer time.

Multiresolution representations have become a key technology for achieving efficient storage together with progressive transmission and visualization performance. Besides that, wavelet-based methods have proven their efficiency for the visualization at diferent levels of detail, progressive transmission and compression of large data sets.

A volume data set is often modeled by a mesh consisting of tetrahedral cells with scalar and/or vector data associated to them. In computer graphics, much research has been devoted to nested tetrahedral meshes generated by recursive decomposition, which are suitable to deal with regularly distributed data points. On the other hand, multiresolution models based on irregular tetrahedral meshes are desirable in order to deal with irregularly-distributed data, since they can accurately capture the shape of the field domain even at the lowest resolution. In this work, we propose an effective 3D model scheme for large volume data that exploits the power of wavelet theory. To achieve this, we present wavelets on tetrahedral domains and we use them as a multiresolution approach that can deal with irregular volume meshes that satisfies the subdivision-connectivity property. That is, we assume that the mesh on which the data is defined can be reached by

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

recursive subdivision of a base mesh and the volumetric data set in encoded as a base mesh (coarse approximation) followed by a sequence of detail coefficients.

The paper is organized as follows: in Section 2 we present an overview of related work on wavelets applied to object modeling. In Section 3 we define Haar-like wavelets over a tetrahedron, we give the filter coefficientes for analysis and synthesis, and we develop an example where a density function defined on a tetrahedron is represented using these waveletes. In Section 4 we present the detailed explanation of wavelet based modeling for irregular volumes, we provide the data structure and we justify why this model allows compression and progressive transmission. Finally, in Section 5 we draw some conclusions and we give an outlook over future work.

2. RELATED WORK

Different methods have been formulated for modeling volumes using tensor products ([Mur00], [Chu00]). The idea of using three dimensional wavelets to approximate three-dimensional volume data sets was introduced in [Mur00], where he constructs a 3D orthonormal wavelet basis using all possible tensor products of one-dimensional basis functions and presents the potential of the 3D wavelet transform (WT) for volume visualization. Although this methodology gives a simple way for constructing wavelets for surface or volume representation; it cannot be used without introducing degeneracies when representing surfaces or volumes defined on general domains of arbitrary topological type, like spherical domains.

Lounsbery [Lou00] and Stollnitz *et al*. [Sto00] were the first who introduced wavelets from a different point of view, defining them on different bounded domains through scaling refinable functions. Refinability, a key property for multiresolution analysis (hereafter referred to as MRA), generalizes the notion of both translation and dilation. Within this context, wavelets defined on arbitrary topological domains on $\Re^2$ are built up in [Lou00]. This approach was generalized *a posteriori* by Sweldens ([Swe00], [Swe01]) who recognized that the *lifting scheme* he proposed was a generalization of Lounsbery's methodology. Other subdivision wavelet construction for functions defined on triangulated spherical domains were introduced by Schröder and Sweldens [Sch00], Nielson *et al.* [Nie00], Bertram *et al.* [Ber00], and Bonneau [Bon00].

In [Lab00] and [Lin00], other techniques for representing volume data are given. However, a tetrahedral mesh can be used to model an object given by sparse data, and this mesh is a general topological domain for the intrinsic representation of the volume. In this case, it is necessary to have wavelets defined over tetrahedra and, based on sucessive refinements, extend the multiresolution analysis to functions defined on them. Hence, beginning with a tetrahedron and using the subdivision as a construction tool, it is possible to generate wavelets on arbitrary topological domains of dimension three or greater. In this paper we present a Haar-like wavelet basis defined over a tetrahedron as the first step for defining this kind of bases over an object represented by tetrahedra.

3. WAVELET BASIS DEFINED OVER A TETRAHEDRON

In order to define the wavelets, it is necessary to adopt a subdivision method. For 2D triangular meshes there are refinement methods satisfying nestedness, consistency and stability. Perhaps the most known one is the combination of red and green refinements proposed by Bank *et al.*, [Ban00]. The stable refinement of tetrahedral meshes is more complex; there is no way a given tetrahedron can be divided into eight congruent ones. However, it is possible to extend the regular 2D refinement strategies mentioned above to 3D. For this purpose different algorithms have been developed during the last years (see, for example, [Bae00], [Mau00], [Bey00]).

Bey [Bey00] introduced a refinement algorithm for unstructured tetrahedral grids, which generates stable and consistent triangulations. In order to do this, he defined some local regular and irregular refinement rules that are applied to single elements. Since our multiresolution model is a regular nested model [Mag00] based on the recursive subdivision of tetrahedra, we choose this method because it is highlighted by the fact that any given tetrahedron results in a group of elements of at most three congruence classes, no matter how many successive refinement steps are performed.

Beginning with a tetrahedral net and using this subdivision method, it is possible to construct wavelets on arbitrary topological domains. To achieve this goal, it is first necessary to define them on a single tetrahedron. In this work we concentrate ourselves on this problem and we provide an example of a L^2 functions representation on the defined bases, as well.

3.1 Haar-like Wavelets

In this section we build a wavelet basis on a tetrahedron following the process given by Girardi and Sweldens [Gir00]. Throughout this section, (*T,*

Σ, μ) is the measure space where T is a tetrahedron with volume V , Σ is the $\sigma-$algebra of all tetrahedra generated by the Bey's subdivision method and μ is the Lebesgue measure. With this procedure we shall obtain Haar-like wavelets on the measure space (T,Σ, μ) that form an unconditional basis for $Lp(T, \Sigma, \mu)$, $1 < p < \infty$.

3.1.1 Wavelets Construction

According to the chosen subdivision scheme, eight new tetrahedra $T_{\beta i}$, $1 \leq i \leq 8$, obtained by subdivision of the father tetrahedron T_α, are introduced during synthesis. This means that these tetrahedra will be generated when going from a resolution level j to a finer resolution level $j + 1$ and will replace the coarser resolution tetrahedron T_α. [Cas00]

Then the refinement equation can be written:

$$\varphi_\alpha = \sum_{\beta \in B(\alpha)} \frac{1}{\sqrt{8}} \varphi_\beta,$$

being $B(\alpha)$ the set of tetrahedra obtained after subdividing the tetrahedron T_α .

The two scale equation for the wavelets is:

$$\psi_\gamma = \sum_\beta g_{\gamma,\beta} \varphi_\beta,$$

being $g_{\gamma,\beta} = \langle \psi_\gamma, \varphi_\beta \rangle$ and $\gamma \in G(\beta)$, $G(\beta)$ the detail tetrahedra obtained when going from resolution $j + 1$to resolution j.

Hence, a single step in the analysis is given by:

$$\varphi_\beta = \frac{1}{\sqrt{8}} \varphi_\alpha(\beta) + \sum_{\gamma \in G(\beta)} g_{\gamma,\beta} \psi_\beta .$$

The generated multiresolution spaces are:

$$V_{k+1} = closspan\{\varphi_\beta, \beta \in B(\alpha)\}$$
$$W_{k+1} = closspan\{\psi_\gamma, \gamma \in G(\beta)\}$$

being

$$V_{k+1} = V_k \oplus W_k .$$

Then, a given function $f \in V_{k+1}$ has two representations:

$$f = \sum_{\beta \in B^*(\alpha)} c_\beta \varphi_\beta,$$

with $B^*(\alpha)$ the set of tetrahedra at the highest resolution and

$$f = \sum_{\alpha \in F_j} c_\alpha \varphi_\alpha + \sum_{\gamma \in G_j} d_\gamma \psi_\gamma ,$$

being F_j the set of tetrahedral at lower resolution levels and G_j the set of details obtained from the decomposition of the mesh until a given level j. As always, c_α and d_γ are found with the fast Haar WT applied to the original signal. (see Sections 3.1.1.1). Then, we must compute these coefficients during the process of analysis and synthesis.

3.1.1.1 Fast Wavelet transform

Analysis

We obtain the low resolution coefficient $c_{j,\alpha}$ from $c_{j+1,\beta}$ with

$$c_{j,\alpha} = \sum_{\forall \beta \in C_j(\alpha)} \frac{1}{\sqrt{8}} c_{j+1,\beta} ,$$

while the details coefficients are:

$$d_{j,\gamma} = \sum_{\beta \in C_j(\alpha)} g_{\gamma,\beta} c_{j+1,\beta} ,$$

where:

$$d_{j,\gamma_1} = \frac{1}{\sqrt{8}} \left(c_{j+1,\beta_1} + c_{j+1,\beta_2} + c_{j+1,\beta_3} + c_{j+1,\beta_4} \right) - \frac{1}{\sqrt{8}} \left(c_{j+1,\beta_5} + c_{j+1,\beta_6} + c_{j+1,\beta_7} + c_{j+1,\beta_8} \right)$$

$$d_{j,\gamma_2} = \frac{1}{2} \left(c_{j+1,\beta_1} + c_{j+1,\beta_2} \right) - \frac{1}{2} \left(c_{j+1,\beta_3} + c_{j+1,\beta_4} \right)$$

$$d_{j,\gamma_3} = \frac{1}{2} \left(c_{j+1,\beta_5} + c_{j+1,\beta_6} \right) - \frac{1}{2} \left(c_{j+1,\beta_7} + c_{j+1,\beta_8} \right)$$

$$d_{j,\gamma_4} = \frac{1}{\sqrt{2}} \left(c_{j+1,\beta_1} - c_{j+1,\beta_2} \right)$$

$$d_{j,\gamma_5} = \frac{1}{\sqrt{2}} \left(c_{j+1,\beta_3} - c_{j+1,\beta_4} \right)$$

$$d_{j,\gamma_6} = \frac{1}{\sqrt{2}} \left(c_{j+1,\beta_5} - c_{j+1,\beta_6} \right)$$

$$d_{j,\gamma_7} = \frac{1}{\sqrt{2}} \left(c_{j+1,\beta_7} - c_{j+1,\beta_8} \right)$$

Synthesis

To recover $c_{j+1,\beta}$ from $c_{j,\alpha}$ and $d_{j,\beta}$, we use:

$$c_{j+1,\beta} := \frac{1}{\sqrt{8}} c_{j,\alpha} + \sum_{\gamma \in G_j(\beta)} g_{\gamma,\beta} d_{j,\lambda} .$$

As the obtained wavelets are orthogonal, the coefficients $g_{\gamma,\beta}$ for the synthesis are equal to those ones used for the analysis.

3.1.1.2 Funtions defined on volumes

We can represent scalar or vector functions defined on volumes using the wavelets defined on a tetrahedron. A scalar function can represent, for example, a density, a temperature, etc. As usual, the fast wavelet transform can be directly applied to the set of coefficients that represents the scalar function. If this function has been mapped to color or texture, the wavelet transform is applied to the mapped function. In Figure 1, we show two steps in the analysis. In this case, the attribute has been mapped to color and the wavelet decomposition has been performed to it.

Figure 1. Wavelet Decomposition of an Attribute defined on a Tetrahedron

4. VOLUME MODELING USING WAVELTS

The mesh representing the object must store the 3D geometry, its topology and its attributes. One of the main advantages of tetrahedral meshes is that any other polyhedral mesh can be reduced to a tetrahedral mesh; hence a tetrahedral mesh can represent a volume with arbitrary topological type. Then, beginning with a tetrahedral mesh and using the subdivision and the defined wavelets, we will show how to generate a model of a volumetric object of arbitrary topological type that can be used for compression and progressive transmission.

4.1 Volumetric Model

The proposed model is based on a semi-regular mesh, which is regular except on the coarsest level tetraheda. This kind of mesh is especially well suited for different multiresolution algorithms. A semi-regular tetrahedral representation is a sequence of approximations at different resolution levels. The corresponding sequence of nested refinements is transformed using the wavelet transform to a representation that consists of a coarse resolution or base mesh and a set of detail coefficients that represents the differences between succesive resolution levels. The base mesh Γ_0 is the mesh at the coarsest resolution and does not have the subdivision-connectivity property. Then, the model consists of a base mesh and a sequence of modifications. These modifications correspond to terms that locally capture the details of the object at different resolutions.

The construction of the multiresolution model starts with a fine tetrahedral mesh Γ_∞ (Figure 2) with the subdivision connectivity property.

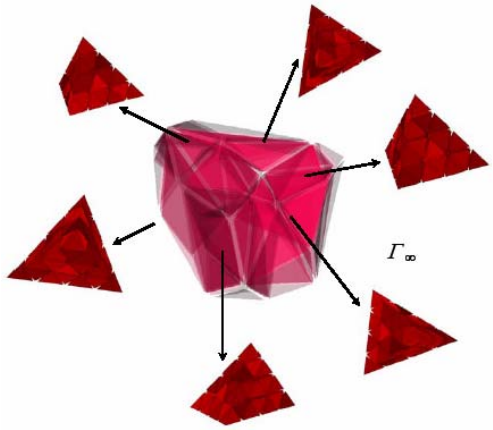


Figure 2. Tetrahedral Mesh

After performing the wavelet transform as many times as possible, we obtain the coarsest resolution mesh Γ_0 plus a set of detail coefficients (Figure 3).

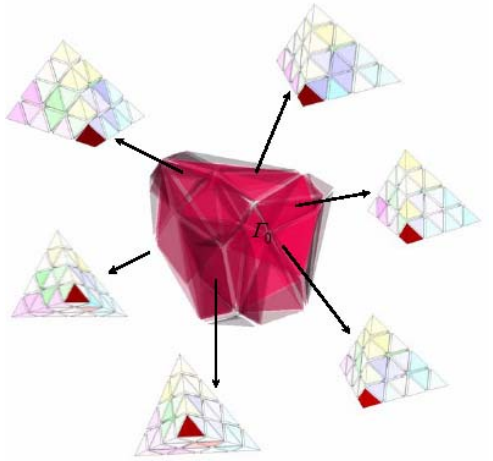


Figure 3. Multiresolution Representation of the Volume

Hence, the developed model begins with the finest resolution mesh Γ_∞ and decomposes it on the coarsest mesh Γ_0, with a set of detail tetrahedra generated during the analysis. This base mesh, together with all the details, are the multiresolution representation of the volume (Figure 3).

Taking into account that the wavelet transform concentrates the energy on the coarsest resolution mesh and that the mesh has space localization, this model is suited for compression. However, the compression will depend not only on the chosen wavelts but also on the following issues:

- The number of coefficients needed to achieve a good approximation of the volume.
- The mesh encoding and storage with the minimun number of bits.

If we transmit the base mesh and all the details, the compression method can be comonsidered as a losless compression encoding. If not all the coefficients are stored, it can be considered a lossy compression encoding. As the highest energy concentration is achieved in the low resolution mesh, only between the 10% and 15% of the detail coefficients are required in order to have a good approximation of the volume

To transmit the underlying mesh of this model *via* Internet, we must consider the transmission of:

- **The base mesh.** A robust transmission has to guarantee that the base mesh is completely transmitted before the details are transmitted and added.
- **The detail coefficients**. After transmitting the base mesh, the details must be sent and added to it.

The transmitted details can be added to the base mesh all at once after they have been received or one by one as long as they are received, until certain requirements are fullfilled. This last option allows the progressive transmission of the model.

4.2 Data Structure

We describe now the data structure proposed for representing the multiresolution volume model. In general, multiresolution meshes admite very compact data structures [Mag00] because the modifications, the involved cells, and the partial order are implicitly defined on the basis of fixed patterns. Then, the data structure for our model must encode the base mesh and the set of details.

The data structure proposed for our model consists on the data structure for the base mesh $\Gamma_0=<\sigma_0, \sigma_1, ..., \sigma_n>$ and on a forest of octrees to store the details corresponding to each one of its cells. Figure 4 shows a tetrahedron of the base mesh and its associated details represented by a forest such its trees have the $d_{0,j}$ as roots. Each tree in the forest is a hierarchy of regular tetrahedra and the relation of dependency is structured as an octree of tetrahedra. As every cell vertex define a regular grid, the coordinates of each vertex can be retrieved from its position in the grid.

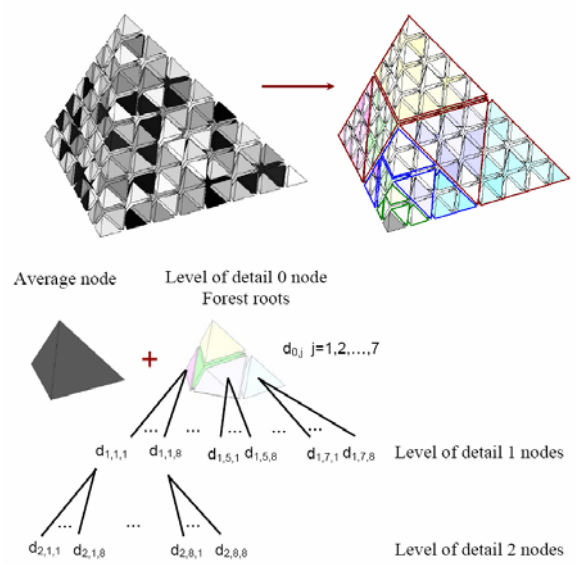


Figure 4. Wavelet Decomposition of a Tetrahedron and graphic Representation of that Decomposition (base mesh+forest)

This multiresolution data structure is generated from a given volume with the conectivity-subdivision property Γ_J, being J the maximum resolution level. The wavelet transform is performed on each set of tetrahedra that replaces the tetrahderon Γ_0 until the lowest resolution tetrahedra Γ_0 is obtained. The set σ_0 coincide with a cell of Γ_0 and its forest of details (Figure 5).

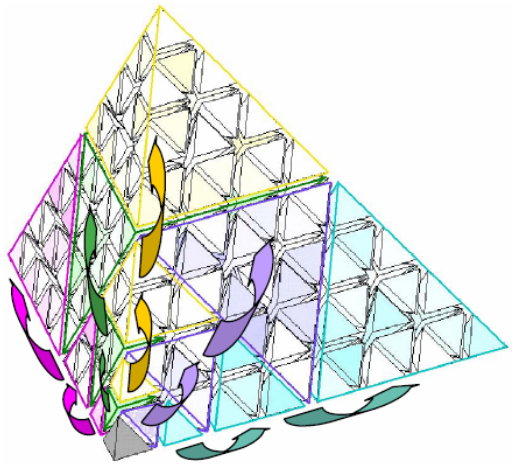


Figure 5. Base Mesh Tetrahedron and the Forest of Details

A forest, the decomposition level and a key to a reference coordinate corresponding to the lowest resolution tetrahedron are stored in a heap for each cell of the base mesh. The reference coordinate will be used to retrieve the geometry of the tetrahedra corresponding to those ones obtained from it at a finer resolution.

4.1.1 Space Complexity of the Data Structure

It is supposed that the mesh is stored using an indexed structure like winged edge, that encodes, for each tetrahedron, the indices of its vertices and the adjacent tetrahedra, along the four faces. The total storage cost corresponding to the data structure of the reference mesh can be calculated in the follong way:

$$TotalCost = ConnectivityCost + GeometryCost + AttributesCost.$$

If n is the number of vertices of the reference mesh and t is the number of tetrahedra, the amount of t tetrahedra is about 6;, the connectivity cost requires to store $4t$ indices (one for each vertex), $4t$ indices corresponding to the adjacent tetrahedra and $3n$ vertex coordinates. Since the cost of an scalar attribute is t, if we consider that we have only one scalar attribute, the storage cost is:

$$TotalCost = 8t + 3n + t = 48n + 3n + 6n.$$

Asuming 4 bytes for the indices, 2 bytes for each coordinate and 2 bytes for a scalar attribute, the storage cost in bytes is $210n$ *bytes*. From this, it is clear that the connectivity information dominates the storage cost and it must be compressed.

The storage cost of our model is:

$$TotalCost = BaseMesh\ Cost + F\ orestCost.$$

The storage cost of the base mesh is the storage cost of a winged edge data structure. So,

$$BaseMesh\ Cost = 210n_{bm}\ bytes,$$

being n_{bm} ($n_{bm} << n$) the amount of vertices of the base mesh. Then:

$$TotalCost = 210n_{bm}\ bytes + F\ orestCost.$$

Each tetrahedron of the base mesh has a forest associated to it and any other node describes a detail tetraedron; the eight sons corresponds to the tetrahedra obtained from Bey's subdivision method.

Each one of the seven trees of the forest is a complete octree that we can implicitly codify; *i.e.* we do not need to store the conectivity and the structural information; since we have regular tetrahedra, the vertex coordinates are implicit. As a consequence, we only need to encode the field or the attribute values in order to encode the details.

We have supposed that each attribute is stored in 2 bytes. These attributes are stored for each tetrahedron of the base mesh and these values have been taken into account in the storage space required for the base mesh. The number of detail tetrahedra plus the tetrahedra of the base mesh is the number of tetrahedra of the reference mesh; however, for each of them we only store the detail that corresponds to the attribute. Then we have $t - t_{bm}$ detail tetrahedra and the storage cost becomes:

$$TotalCost = 204n_{bm}\ bytes + t - t_{bm}$$
$$= 210n_{bm}\ bytes + (6n - 6n_{bm})byte.$$

The storage cost is significantly reduced compared to the total cost of the reference mesh ($n_{bm} << n$).

4.3 Example

It is worth to notice that when the size of the base mesh compared to the size of the highest resolution one gets smaller, the storage cost diminishes. Table 1 shows a comparison of the storage cost between the reference mesh and our approach at n different resolutions.

	Cost (in bytes)			
n	Reference Mesh	Base Mesh	Forest	Our approach
8	603979776	75497472	14680064	90177536
7	75497472	9437184	1835008	11272192
6	9437184	1179648	229376	1409024
5	1179648	147456	28672	176128
4	147456	18432	3584	22016
3	18432	2304	448	2752
2	2304	288	56	344
1	288	36	7	43

Table 1

5. MESH COMPRESSION AND PROGRESSIVE TRANSMISSION

We give now the structure to represent the complete multiresolution mesh. However, this wavelet based representation can be compressed even more in orden to reduce storage space and transmision time. We will see what elements must be introduced in the structure to achieve higher levels of compresion. Besides, as we want embedded encoding for progressive transmission, we will see how to include it in the structure.

5.1 How the Model allows Compression

We will apply the compression technique to a scalar function defined on the tetrahedra and we will show how the cell based scheme allows to achieve a high compression level. We can have two different alternatives to compress the volume: one is to reduce the number of coefficients to approximate the volumetric data and the other is to encode and to store the necessary information using a small number of bits.

5.1.1 Two models for Compression

We suppose to have the multiresolution model as

described in the above sections and that the WT was performed to the given data. Afterwards, the attribute values associated to the tetrahedra are decorrelated and the energy of the original data is concentrated in a relative small number of coefficients.

The key behind wavelet compression is to select the coefficients with smallest norm and replace them by zero. This criterion minimizes the L^2 norm of the resulting approximation error. Whatever is the selected criteria to set the detail coefficients to zero, the original signal will be approximated with a very small number of nonzero coefficients. Then, we can obtain compression in two different ways:

- All coefficients that remain in the representation are encoded with a lower amount of bits per coefficient using run-length encoding, vectorial quantization or differential encoding.
- A very small portion of coefficients (between 10% and 15%, for example) are kept and the rest of them are set to zero. Hence, it would be reasonable to keep only the non zero coefficients. In order to do this, we can take advantage of the spatial localization property of the wavelets: the behaviour of the detail coefficients of a father in a given forest tree allows to predict the behaviour of its descendents.

5.1.2 Error Control

In the models previously described we have supposed that we could control the level of compression by specifying a given percentage of coefficents that are not set to zero (surviving coefficients). In both cases, we can control the number of surviving coeffients by specifying an adequate treshold and set to zero all coefficients whose magnitudes are smaller than it. This threshold can be automatically determined taking into account the maximum allowed approximation error. The ideal way to compute the threshold is by sorting all the coefficients in decresing order of significance. However, when the amount of data is huge, this is impractical ($O(n \log n)$). Then, if we want to control the error, we should find the threshold without sorting the data.

One can also specify a threshold and encode all the coefficients of magnitude greater than it and eliminate all the other ones ($O(n)$). In this case, even if the approximation error can be computed, it can not be controled since a fixed number of coefficients are eliminated, depending on their value respect to the threshold.

Finally, the compression scheme developed so far allows compression of non structured volumes decomposed in atomic tetrahedral elements and that have scalar or vectorial values defined on them. It is then necessary to consider the appropiate metric depending on the nature of data, *e.g.* geometric, color, texture data, etc. in general the L^2-norm is considered.

5.1.3 Decompression

The decompression allows to reconstruct the received information of the progressive transmission. The base mesh will be received first and will be decoded according to the encoding method. Once the reconstruction of the base mesh is completed, the inverse WT will be applied to the detail coefficients received afterwards.

5.2 How the Model allows Progressive Transmission

For the mesh transmision we use a modification of the protocol for the transmission of semi-regular meshes, the so-called *mesh transfer protocol (MTP)* defined by Staadt [Sta00]. In order to guarantee a reliable and ordered delivery of the base mesh to destination, we use TCP for the transmission. To do this, the protocol must use a lot of overhead communication; fortunately the base mesh is small compared to the finest resolution level of the mesh.

After the transmission of the base mesh has been completed, the details must be sent. In this case, the protocol used to send them will depend on the implemented model:

- If the details are sent without position information it means that it is implicitely given by the order the chain is transmitted and TCP must be used for detail transmission.
- If each detail record contains the complete topological information that specifies the tetrahedron to which the detail has to be added, the ordering of the detail records is not important. For the reconstruction of the original mesh it is even not necessary that every detail record is received: the loss of records may result only in small local errors. Therefore, it is possible to use the UDP protocol for their transmission.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have extended the definition of the wavelets defined on a single tetrahedron to tetrahedral preserving the MRA. Based on them it was shown that it is possible to represent irregular tetrahedral meshes using a multiresolution approach. We have also considered only scalar functions defined on tetrahedra since they represent the volume attributes. The proposed model allows compression

and progressive transmission. In compression, we showed how to reduce the number of coeffients needed for approximating the volumetric data and how to encode the information according to the proposed models. In the case of transmission, we analyzed a protocol that allows the progressive transmission of the mesh.

We consider that the described framework is an important initial work to construct multiresolution representations of irregular meshes. Future work includes the extension of our results to functions defined on unstructured tetrahedral domains and also the representation of the underlying domain geometries. These extensions would allow to obtain a wavelet-based method to model irregular tetrahedral meshes without the subdivision connectivity property. The used oracles and reconstruction schemes are based on the L^2 norm. Furthermore, we are studying the possibility of taking into account other error criteria based on human perception in order to optimize the approximation quality.

7. ACKNOWLEDGMENTS

This work was partially supported by grant 24/N015 (SECyT, UNS).

8. REFERENCES

[Bae00] Baensch, E. Local Mesh Refinement in 2 and 3 Dimensions. Impact of Computing in Science and Engineering, pp.181-191, 1991.

[Ban00] Bank, R. E., *et al*. Refinement Algorithms and Data Structures for Regular Local Mesh Refinement. Scientific Computing, pp.3-17, 1983.

[Ber00] Bertram, M., *et al*. Bicubic Subdivision-surface Wavelets for large Scale Isosurface Representation and Visualization. ACM Proc. SIGGRAPH, pp. 389-396, 2000.

[Bey00] Bey, J. Tetrahedral Grid Refinement. Computing, pp.355-378, 1995.

[Bon00] Bonneau, G.P. Multiresolution Analysis on Irregular Surface Meshes. IEEE Trans. Visualization and Computer Graphics, vol.4, pp. 365-378, 1998.

[Cas00] Castro, S., Castro, L., Boscardín, L. & De Giusti, A. Wavelet Representation of Functions defined on Tetrahedrical Grids. Journal of Computer Science & Technology, Vol.2, Nº 6, pp. 30-42, 2002.

[Chu00] Chui, C. An Introduction to Wavelets. Charles Chui Series Ed., Academic Press, 1992.

[Cig00] Cignoni, P., *et al*. Multiresolution Modeling and Rendering of Volume Data based on Simplicial Complexes. Proc. of Symposium on Volume Visualization, pp. 19-26, 1994.

[Gro00] Gross, M. L^2 Optimal Oracles and Compression Strategies for Semi-Orthogonal Wavelets. TR 254, Computer Science Department, ETH Zürich, 1996.

[Lab00] Labsik, U. Multiresolution Mesh Processing for Triangular and Tetrahedral Meshes. PhD dissertation, Der Technischen Fakultät der Universität Erlangen-Nürnberg, 2003.

[Lin00]Linsen, L., *et al*. Hierarchical Large-Scale Volume Representation with $\sqrt[3]{2}$ Subdivision and Trivariate B-Spline Wavelets. TR CSE-2002-7, Department of Computer Science, University of California, Davis, 2002.

[Lou00] Lounsbery, J. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. PhD dissertation, University of Washington, Washington, Seattle, 1994.

[Mag00] Magillo, P., De Floriani, L. Multiresolution Mesh Representation: Models and Data Structures. In Tutorial on Multiresolution in Geometric Modelling, Spriner Verlag, pp.363-416, 2002.

[Mau00] Maubach, J. M. Local Bisection Refinement for N-Simplicial Grids Generated by Reflection. SIAM J. Sci. Computing 16, pp.210-227, 1995.

[Mur00] Muraki, Sh. Approximation and Rendering of Volume Data using Wavelet Transforms. Proc. of IEEE Visualization, pp. 21-28, 1992.

[Nie00] Nielson, G., *et al*. Haar Wavelets over Triangular Domains with Applications to Multiresolution Models for Flow over a Sphere. Proc. of IEEE Visualization, pp. 143-150, 1997.

[Pas00] Pascucci, V., Coord. Multiresolution Modeling, Visualization and Compression of Volumetric Data. Notes for the 3rd Tutorial of the IEEE Conference on Visualization, October 2003.

[Sch00] Schröeder, P., Sweldens, W. Spherical Wavelets: Efficiently representing Functions on the Sphere. ACM Proc. SIGGRAPH, pp. 161-172, 1995.

[Sta00] Staadt, O. Multiresolution Representation and Compression of Surfaces and Volumes. PhD dissertation, Swiss Federal Institute of Technology Zürich, 2001.

[Sto00] Stollnitz, E., *et al*. Wavelets for Computer Graphics: Theory and Applications. Morgan & Kaufmann Publishers, Inc, 1996.

[Swe00] Sweldens, W. The Lifting Scheme: A new Philosophy in Biorthogonal Wavelet Constructions. Proceedings of the SPIE 2569, pp.68-79, 1995.

[Swe01] Sweldens, W. The Lifting Scheme: A Custom-design of Biorthogonal Wavelets. Applied and Computational Harmonic Analysis 3, pp.186-200, 1996.

Fast Near Phong-Quality Software Shading

Tony Barrera
Barrera Kristiansen AB
Uppsala,Sweden
tony.barrera@spray.se

Anders Hast
Creative Media Lab
University of Gävle, Sweden
aht@hig.se

Ewert Bengtsson
Centre For Image Analysis
Uppsala University, Sweden
ewert@cb.uu.se

ABSTRACT

Quadratic shading has been proposed as a technique giving better results than Gouraud shading, but which is substantially faster than Phong shading. Several techniques for fitting a second order surface to six points have been proposed. We show in this paper how an approximation of the mid-edge samples can be done in a very efficient way. An approximation of the mid-edge vectors are derived. Several advantages are apparent when these vectors are put into the original formulation. First of all it will only depend on the vertex vectors. Moreover, it will simplify the setup and no extra square roots are necessary for normalizing the mid-edge vectors. The setup will be about three times faster than previous approaches. This makes quadratic shading very fast for interpolation of both diffuse and specular light, which will make it suitable for near Phong quality software renderings.

Keywords

Quadratic shading, Phong shading.

1. INTRODUCTION

Quadratic shading has been proposed as a technique giving better results than Gouraud shading [Gour71], but which is substantially faster than Phong shading [Phon75]. Quadratic interpolation could be setup by fitting a second order surface to six points. Both the diffuse and specular light must be computed at these points and the best quality is obtained if these are interpolated separately. Furthermore, the power in the specular computation must be computed per pixel. Besides this computation, quadratic shading, including both diffuse and specular light, can be performed using four additions per pixel instead of four additions and a reciprocal square root for Phong shading.

The main drawback using quadratic shading has been the rather complex rasterization setup. This paper proposes a solution to this problem and the simplified setup presented in this paper will be about three times faster than previous setup approaches. Hence, quadratic shading will be suitable for software rendering and could therefore be implemented on hand held devices like cell phones etc.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

2. PREVIOUS WORK

A 3D object represented by polygons will appear faceted when rendered, unless some kind of shading technique is used that interpolates intensities or colors over the polygons. Gouraud uses bilinear interpolation of the colors at the vertices. If the intensities are interpolated, then only one addition per pixel is necessary to achieve this type of shading.

The intensity over the polygon in screen space can be considered as a plane in (x, y, Φ)-space, where x and y are the screen coordinates, and Φ is the intensity. Phong uses bilinear interpolation of the normals at the edges, to obtain intermediate normals for each pixel. These normals are then used in the illumination computation. Duff [Duff79] shows how the computation can be implemented in an efficient way, requiring only three additions, one division and one square root per pixel for Phong shading. The intensity will form a smooth surface in (x, y, Φ)-space. Phong shading produces better results than Gouraud. Nonetheless, Phong does not produce correct shading, in the sense that Phong shading is also an approximation of the 'real' shading curve. This can easily be proved by the fact that the Phong intensity curve is not necessarily C^1 continuous over polygon edges.

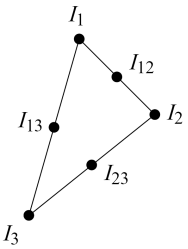


Figure 1: *Sample points for quadratic shading, where the mid edge samples are I_{12}, I_{13} and I_{23}.*

Therefore, an approximation of the intensity curve will suffice. Bishop and Weimer [Bish86] use a Taylor series expansion of Duff's equation to obtain a second order bi-variate polynomial. It could be evaluated by only two additions per pixel along a scanline. This quadratic interpolation scheme will produce a second order polynomial surface in (x, y, Φ)-space.

Kappel [Kapp95] uses a surface fitting technique based on Powell-Sabin [Powe77] quadratic interpolation using the Cendes-Wong [Cend87] formulae. The purpose of that approach is to eliminate Mach bands by generating a C^1 continuous surface over the polygon. This is possible by subdividing the triangle into smaller triangles. Kirk et al. [Kirk92] solves this problem by fitting a surface to a triangle, where the intensity of the vertices of adjacent triangles are the same, and the first derivative of the intensity at the edges tend toward being continuous.

Quadratic shading

Kirk and Voorhies [Kirk90] adopt another approach to quadratic shading which is somewhat different from previous approaches. They show that quadratic interpolation could be setup by fitting a second order surface to six points, yielding a polynomial with six unknown coefficients, which must be determined. The sample points are the vertices and edge mid points of a triangle, as shown in Fig 1.

We will throughout this paper assume that a parallel light source is used, as well as a constant view vector. However, it will be discussed how a point light source can be used and what complications it will give for the algorithm. A parallel light source is often used when speed is crucial, as it usually is for software renderings on hand held devices. The light vector must be interpolated over the polygon using Phong shading when point light sources are used. However, for quadratic shading it is possible to use the light source vectors at the six sample points instead. Furthermore, the view vector should be interpolated over the polygon for Phong shading. However, a simplification often made when speed is crucial is to use a constant view vector, i.e. the viewer is assumed to be at infinity. With a distant light source the six samples are computed as

$$I_1 = \mathbf{L} \cdot \mathbf{N}_1 \tag{1}$$

$$I_2 = \mathbf{L} \cdot \mathbf{N}_2 \tag{2}$$

$$I_3 = \mathbf{L} \cdot \mathbf{N}_3 \tag{3}$$

$$I_{12} = \mathbf{L} \cdot \frac{\mathbf{N}_1 + \mathbf{N}_2}{\| \mathbf{N}_1 + \mathbf{N}_2 \|} = \frac{I_1 + I_2}{\| \mathbf{N}_1 + \mathbf{N}_2 \|} \tag{4}$$

$$I_{13} = \mathbf{L} \cdot \frac{\mathbf{N}_1 + \mathbf{N}_3}{\| \mathbf{N}_1 + \mathbf{N}_3 \|} = \frac{I_1 + I_3}{\| \mathbf{N}_1 + \mathbf{N}_3 \|} \tag{5}$$

$$I_{23} = \mathbf{L} \cdot \frac{\mathbf{N}_2 + \mathbf{N}_3}{\| \mathbf{N}_2 + \mathbf{N}_3 \|} = \frac{I_2 + I_3}{\| \mathbf{N}_2 + \mathbf{N}_3 \|} \tag{6}$$

A second order polynomial is constructed using these six samples and the coordinates of the six point. Such a quadratic shading function is defined by

$$\Phi(x,y) = Ax^2 + By^2 + Cxy + Dx + Ey + F \tag{7}$$

Kirk and Voorhies do not show how these coefficients are computed, neither do Saxe et al. [Saxe96] who use this type of quadratic interpolation in Pixel-Planes 5 to display radiosity illuminated models [Fuch89].

Seiler [Seil98] proposes a simple and fast way to setup the coefficients for this type of quadratic shading. How the computation can be done is also shown by Abbas et al. [Abba01a, Abba01b] but they do not try to make the computation as efficient as Seiler does. Both methods do not cover special cases which will lead to division by zero. This problem however, is solved by Lee and Jen [Lee01] who have simplified Seiler's approach. In [Hast01] it is shown how these methods can be simplified further in order to make the computation of the coefficients as efficient as possible.

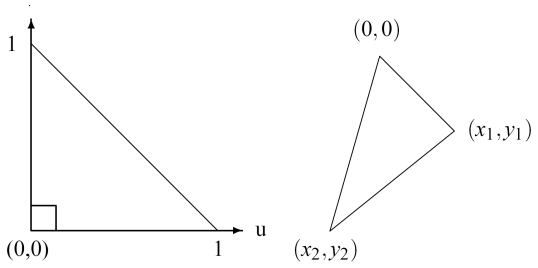


Figure 2: *To the left: Ortho-normalised triangle. To the right: a polygon with relative vertices*

In general the coefficients can be obtained by setting up a system of equations using equation (7) and the relative coordinates of the sample points. Relative coordinates are preferable since it will yield a system which is easier to solve, as it will contain more zeroes. The relative coordinates are obtained by

shifting the triangle so that the top vertex has coordinate (0,0). The middle vertex will have coordinate (x_1, y_1), and the bottom most vertex will be (x_2, y_2), as shown to the right in figure 2. This implies that the vertices have to be sorted in the y direction. Note that, these operations are done in screen space and it is here presumed that the top left corner is coordinate (0,0). The system of equations is

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ x_2^2 & y_2^2 & x_2 y_2 & x_2 & y_2 & 1 \\ \left(\frac{x_1}{2}\right)^2 & \left(\frac{y_1}{2}\right)^2 & \frac{x_1}{2}\frac{y_1}{2} & \frac{x_1}{2} & \frac{y_1}{2} & 1 \\ \left(\frac{x_2}{2}\right)^2 & \left(\frac{y_2}{2}\right)^2 & \frac{x_2}{2}\frac{y_2}{2} & \frac{x_2}{2} & \frac{y_2}{2} & 1 \\ \left(\frac{x_1+x_2}{2}\right)^2 & \left(\frac{y_1+y_2}{2}\right)^2 & \frac{x_1+x_2}{2}\frac{y_1+y_2}{2} & \frac{x_1+x_2}{2} & \frac{y_1+y_2}{2} & 1 \end{bmatrix} \quad (8)$$

And the problem is to solve this equation

$$M\mathbf{c} = \Phi \quad (9)$$

where

$$\Phi = [I_1, I_2, I_3, I_{12}, I_{13}, I_{23}]^T \quad (10)$$

$$\mathbf{c} = [A, B, C, D, E, F]^T \quad (11)$$

2.1.1 Alternative derivation

An alternative and ultimately slightly faster approach is to use an ortho-normalized triangle with p_0 = (0,0), p_1 =(1,0) and p_2 = (0,1), as shown to the left in figure 2. It has been obtained by first sorting the vertices, and then shifting them as shown in the same figure to the right. The coefficients for a bi-variate polynomial over this triangle is first computed by setting up the following system of equations

$$\Phi = M\mathbf{c} \quad (12)$$

where

$$\Phi = [I_1, I_2, I_3, I_{12}, I_{13}, I_{23}]^T \quad (13)$$

$$\mathbf{c} = [a, b, c, d, e, f]^T \quad (14)$$

And

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 1 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix} \quad (15)$$

The solution for $\mathbf{c} = M^{-1}\Phi$ is

$$\begin{aligned} a &= 2(I_1 + I_2 - 2I_{12}) \\ b &= 2(I_1 + I_3 - 2I_{13}) \\ c &= 4(I_1 + I_{23} - I_{13} - I_{12}) \\ d &= I_2 - I_1 - a \\ e &= I_3 - I_1 - b \\ f &= I_1 \end{aligned} \quad (16)$$

where

$$\Phi(u, v) = au^2 + bv^2 + cuv + du + ev + f \quad (17)$$

It is shown in [Hast03a] and [Hast02b] how the setup variables needed for incrementally stepping along the edges as well as along the scanlines, can be computed directly using equation 17 instead of using equation 7. This approach is actually slightly faster. It was also shown in [Hast02a] how this could be used for bump mapping. The reason why the ortho-normalized triangle is used is that the coordinates for the vertices are (0,0), (1,0) and (0,1) and this yields a much more simple matrix inverse to solve, i.e the inverse of equation (15). The three different solutions discussed in [Hast01] and originally developed by Lee & Jen [Lee01], Seiler [Seil98] and Abbas et al. [Abba01a, Abba01b] are actually three approaches for solving this inverse. The inverse of this new matrix formulation becomes much more straight-forward to solve. Next a transformation is needed from the actual triangle (the triangle to the right in figure 2) into the so called ortho-normalized triangle. For any point (x, y) on the actual triangle, a corresponding point (u, v) on the normalized triangle can be obtained by the following transformation

$$u = x\alpha_u + y\beta_u \quad (18)$$

$$v = x\alpha_v + y\beta_v \quad (19)$$

where

$$\begin{aligned} \alpha_u &= y_2\omega \\ \beta_u &= -x_2\omega \\ \alpha_v &= -y_1\omega \\ \beta_v &= x_1\omega \end{aligned} \quad (20)$$

and

$$\omega = \frac{1}{x_1 y_2 - x_2 y_1} \quad (21)$$

Finally equations (18) and (19) are put into equation (17) and the terms are rearranged so that we obtain the coefficients for (7). This is done by extracting the terms multiplied by x^2, x and so forth. The coefficients are

$$
\begin{aligned}
A &= a\alpha_u^2 + b\alpha_v^2 + c\alpha_u\alpha_v \\
B &= a\beta_u^2 + b\beta_v^2 + c\beta_u\beta_v \\
C &= 2a\alpha_u\beta_u + 2b\alpha_v\beta_v + c(\alpha_u\beta_v + \alpha_v\beta_u) \\
D &= d\alpha_u + ea\alpha_v \\
E &= d\beta_u + e\beta_v \\
F &= f
\end{aligned} \qquad (22)
$$

The time needed for computing these coefficients as well as the computation time for computing them using the previously explained approaches will be shown later in this paper.

3. APPROXIMATION OF MID-EDGE VECTORS

One of the drawbacks with the quadratic setup is the computation and normalization of the mid-edge vectors that are needed in order to compute the intensities I_{12}, I_{13} and I_{23} as equations (4) through (6) show. We will now show that these computations can be avoided and the setup will end up being even simpler but also faster. Our working name for this approach has been X-shading and we will therefore refer to it as that in the subsequent sections. An approximation based on one step of the Newton-Raphson method was used by Wynn [Wynn01] for normalization of an interpolated normal

$$\mathbf{N}' = \frac{\mathbf{N}}{2}(3 - \mathbf{N}\cdot\mathbf{N}) \qquad (23)$$

We shall derive an equation for computing the approximation of a normalized vector that is halfway in between two vectors. The vector in between is computed as

$$\mathbf{N}_{1/2} = \frac{\mathbf{N}_1 + \mathbf{N}_2}{2} \qquad (24)$$

Use equation (23) to normalize equation (24), by letting $\mathbf{N} = \mathbf{N}_{1/2}$, in the following way

$$\mathbf{N}'_{1/2} = \frac{\mathbf{N}_1 + \mathbf{N}_2}{4}\left(3 - \frac{\mathbf{N}_1 + \mathbf{N}_2}{2}\cdot\frac{\mathbf{N}_1 + \mathbf{N}_2}{2}\right) \qquad (25)$$

Expand the equation and note that the normals are normalized and therefore $\mathbf{N}_1\cdot\mathbf{N}_1 = 1$ and $\mathbf{N}_2\cdot\mathbf{N}_2 = 1$. Hence

$$\mathbf{N}'_{1/2} = \frac{\mathbf{N}_1 + \mathbf{N}_2}{4}\left(3 - \frac{2(\mathbf{N}_1\cdot\mathbf{N}_2) + 2}{4}\right) \qquad (26)$$

And after rearranging the terms

$$\mathbf{N}'_{1/2} = (\mathbf{N}_1 + \mathbf{N}_2)\frac{5 - \mathbf{N}_1\cdot\mathbf{N}_2}{8} \qquad (27)$$

Accuracy of the proposed approximation

This approximation of the mid-edge vectors works quite well for normals with an angle that are less than 45° as shown in figure 3 where the vector norm of the mid-edge vector is computed for different angles. The norm is very close to one for angles less than 45°. Then the norm starts to decrease rather rapidly, as shown in the figure, but it should be noted that the scale goes from 0.88 to 1.0. Hence, the approximation is still not so bad for angles over 45°.

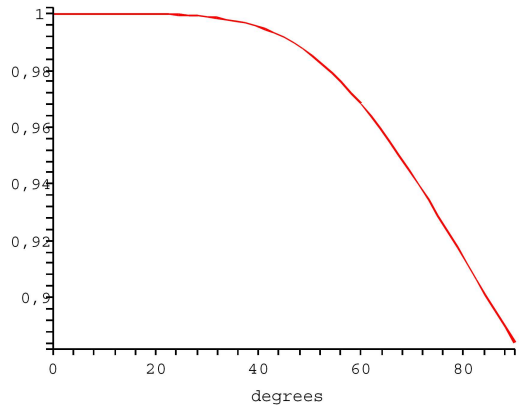


Figure 3: *The vector norm of the approximation for different angles between the vectors.*

The approximation will always do better than a linear approximation, which assures that the quadratic approach will always be better than Gouraud shading. However, the quality of the highlight will be affected for larger angles. It could be argued though, that such large angles means that the polygons should be subdivided further.

4. FASTER QUADRATIC SETUP

In the next step we use this approximation for computing the mid-edge vectors and then these are substituted into equations (16).
The intensities are computed as

$$I_{12} = \mathbf{L}\cdot\mathbf{N}'_{1/2} = \mathbf{L}\cdot(\mathbf{N}_1 + \mathbf{N}_2)\frac{5 - \mathbf{N}_1\cdot\mathbf{N}_2}{8} \qquad (28)$$

Hence

$$I_{12} = (I_1 + I_2)\frac{5 - \mathbf{N}_1\cdot\mathbf{N}_2}{8} \qquad (29)$$

Similarly

$$I_{13} = (I_1 + I_3)\frac{5 - \mathbf{N}_1\cdot\mathbf{N}_3}{8} \qquad (30)$$

$$I_{23} = (I_2 + I_3)\frac{5 - \mathbf{N}_2\cdot\mathbf{N}_3}{8} \qquad (31)$$

Use equations (29) through (31) and substitute them into equations (16). After simplification, the solution for X-shading now becomes

$$
\begin{aligned}
a &= (I_1 + I_2)(\mathbf{N}_1 \cdot \mathbf{N}_2 - 1)/2 \\
b &= (I_1 + I_3)(\mathbf{N}_1 \cdot \mathbf{N}_3 - 1)/2 \\
c &= a + b - (I_2 + I_3)(\mathbf{N}_2 \cdot \mathbf{N}_3 - 1)/2 \\
d &= I_2 - I_1 - a \\
e &= I_3 - I_1 - b \\
f &= I_1
\end{aligned}
\tag{32}
$$

It should be mentioned that if a point light source was used instead of a parallel light source, then we for an example would obtain

$$
I_{12} = \mathbf{L}_{12} \cdot (\mathbf{N}_1 + \mathbf{N}_2) \frac{5 - \mathbf{N}_1 \cdot \mathbf{N}_2}{8} \tag{33}
$$

where $\mathbf{L}_{12}$ is the light source vector in the direction to the point light source at the sample point in question. This will make the setup a bit more complicated. Nonetheless, the normalization of the mid-edge vector is still avoided.

Note that this solution only depends on the vertex normals and the reason for this is of course that the approximation of the mid-edge vectors only depends on the vertex normals. The effect of this is that the extra normalization of the mid-edge vectors, which are shown in (4) through (6), disappears. Hence, a substantial amount of computation time is saved by using the proposed method, especially on systems where there is no hardware for computing divisions and square roots.

Lee & Jen	Abbas et al.	Seiler	Hast et al.	X-shading
40.6	42.0	49.4	40.3	12.7

Table 1: *Timing in seconds of one hundred million computations of the coefficients on an 1.8 GHz Pentium 4.*

Table 1 shows the timing for the computation of the coefficients for equation (7). The earlier mentioned methods proposed by Lee and Jen, Siler and Abbas et al, and the fast version proposed by Hast et al. in [Hast02b, Hast03a] and the method proposed in this paper for one hundred million computations.

The new method is about three times faster than previous methods. It should be mentioned that the approximation of the mid-edge vectors could be used for all the other previously mentioned approaches as well. However, we chose to use it for the fastest approach in this paper. Another important thing to mention is that the dot products in equations (32) could be precomputed and stored since the angle between the vertex normals are invariant to affine transformations such as translation and rotation. In fact ($\mathbf{N}1 \cdot \mathbf{N}2 - 1)/2$ and so forth could be precomputed and stored. This will make the proposed approach even faster.

Figure 4: *A Phong shaded Venus de Milo statue.*

Figure 5: *A shaded Venus de Milo statue using the proposed approach*

5. HIGH QUALITY HIGHLIGHTS

Both the diffuse and specular light can be interpolated using the proposed X-shading approach. When near Phong quality highlights are required for large polygons, it is necessary to do one interpolation for each type of light. The diffuse light is generally computed as

$$I_d = \mathbf{N} \cdot \mathbf{L} \qquad (34)$$

where **N** is the normal and **L** is the vector in the direction to the light source. We omit the color of the surface and the color of the light source for simplicity. The intensities computed for each vertex using this equation can be interpolated by the proposed approach. Likewise, the specular intensity can be interpolated separately by computing the specular intensity for each vertex by

$$I_s = \mathbf{N} \cdot \mathbf{H} \qquad (35)$$

where **H** is the halfway vector introduced by Blinn [Blin77]. Note, that this is not the full specular computation. For each pixel it is necessary to compute $(I_s)^s$ where s is the so called shininess. A faster technique is proposed in [Hast03b] for computing the specular light, which will be very well suited for software rendering. The fact that two different interpolations are used does not make the whole setup twice as large. Several intermediate variables needed in the first setup for the diffuse light can be reused for the second setup for the specular light. Anyhow, it will be faster than a true Phong interpolation, which will need two bilinear interpolations, one bi-quadratic interpolation, as well as one division and one square root per pixel, in order to compute the diffuse and specular light. Figure 4 shows the back of the famous Venus de Milo statue that has been shaded by Phong shading using a polygon setup approach. This image can be compared to the same object in figure 5 shaded by the proposed X-shading approach. The difference is hardly noticeable.

A torus with only 288 triangles where every two adjacent triangles constitute a planar surface, was shaded in figure 6 and 7. The maximum angle between two normals are as high as 41.4° and obviously it should have been subdivided further since the contour is not looking good. Anyway, the shading and especially the highlights are indistinguishable from each other. The conclusion is that X-shading, even though an approximation is used for the mid-edge vectors, still will produce satisfactory results. Moreover, it is much faster than Phong shading and even faster than previous quadratic shading approaches. This will make X-shading attractive for software rendering. Especially since several divisions and all the square roots needed for computing the coefficients in the bi-variate quadratic equation have been removed from the setup.

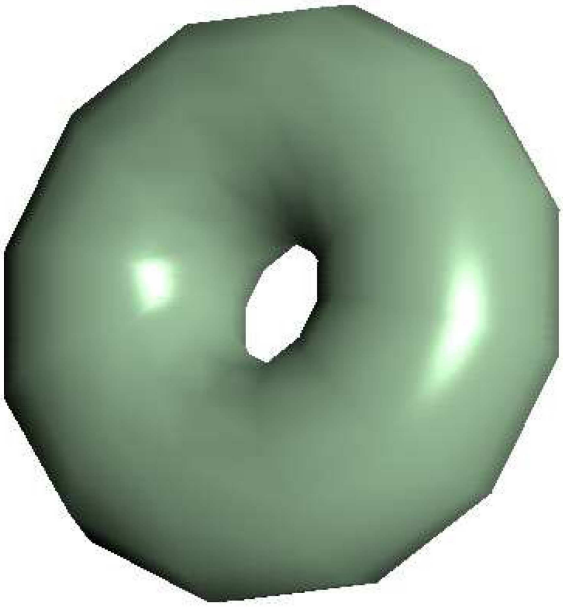


Figure 6: *A Phong shaded torus with 288 polygons.*

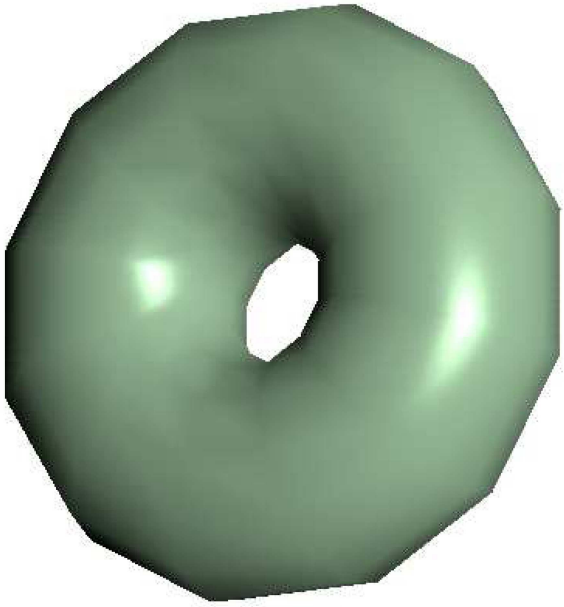


Figure 7: *A shaded 288 polygon torus using the proposed approach*

6. CONCLUSIONS

The proposed setup approach for quadratic shading is about three times faster than previous approaches. The approximation is accurate enough for normals where the angle is less than 45°, and will still do better than Gouraud shading for larger angles. Since the angles are usually smaller than that, otherwise the contour of the object will look far from smooth, the proposed method will produce near Phong quality highlights without any square roots and only one

division needed for computing the coefficients in the bi-variate quadratic equation. This can be compared with the other approaches that need at least three more divisions and square roots as implied by equations (4) through (6). Thus, the proposed approach is very well suited for software rendering where speed is crucial.

Today, graphics hardware makes Phong shading useful for 3D applications like games and real-time visualizations. However, it is still out of reach for many types of hand held devices. It is reasonable to presume that such devices will include more and more applications using graphics. Any graphics card on such a device will be much simpler than the card used for gaming on a home computer since graphics cards are big energy consumers and energy is crucial for these types of devices. Even though some attempts to construct energy saving graphics cards have been done [Kame03], it is reasonable to presume that software rendering will be used for 3D graphics on many of these devices for a long time ahead.

As future work we are planning to investigate how X-shading would compare to Gouraud shading on hand held devices such as those on a cell phone. The small display area versus the limited processing power leads to some interesting trade-offs.

7. REFERENCES

[Abba01a] A. M. Abbas, L. Szirmay-Kalos, G. Szijarto, T. Horvath, T. Foris *Quadratic Interpolation in Hardware Rendering,* Spring Conference of Computer Graphics, 2001.

[Abba01b] A. M. Abbas, L. Szirmay-Kalos, T. Horvath, T. Foris *Quadratic Shading and its Hardware Implementation*, Machine Graphics and Vision, Vol. 9, No. 4, pp. 825-804, 2001.

[Bish86] G. Bishop, D. M. Weimer, *Fast Phong Shading* Computer Graphics, vol. 20, No 4, pp. 103-106, 1986.

[Blin77] J. F. Blinn, *Models of Light Reflection for Computer Synthesized Pictures,* In Proceedings SIGGRAPH, pp. 192-198. 1977.

[Cend87] Z. J. Cendes, S. H. Wong, *C^1 Quadratic Interpolation over Arbitrary point sets,* Computer Graphics & Applications, Vol. 7, No. 11, pp. 8-16, 1987.

[Duff79] T. Duff, *Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays* ACM, Computer Graphics, Vol. 13, pp. 270-275, 1979.

[Fuch89] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, L. Israel, *A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories* Computer Graphics (Proc. of SIGGRAPH '89), Vol. 23, No. 3, pp 79- 88.

[Gour71] H. Gouraud, *Continuous Shading of Curved Surfaces,* IEEE transactions on computers vol. c-20, No 6, June 1971.

[Hast01] A. Hast, T. Barrera, E. Bengtsson, *Faster Computer Graphics - by Reformulation and Simplification of Mathematical Formulas and Algorithms*, IMAGINE2001/F.E.S.T, 2001.

[Hast02a] A. Hast, T. Barrera, E. Bengtsson, *Improved Bump Mapping by using Quadratic Vector Interpolation*, Eurographics02, short paper/Poster 2002.

[Hast02b] Anders Hast, *Licentiate Thesis: Improved Fundamental Algorithms for fast Computer Graphics*, 2002.

[Hast03a] A. Hast, T. Barrera, E. Bengtsson *Fast Setup for Bilinear and Biquadratic Interpolation over Triangles,* Graphics Programming Methods, Charles River Media, pp. 299-314, 2003.

[Hast03b] A. Hast, T. Barrera, E. Bengtsson, *Fast Specular Highlights by modifying the Phong-Blinn Model,* A. Hast,T. Barrera, E. Bengtsson, Siggraph, Sketches and applications 2003.

[Kapp95] M. R. Kappel, *Shading: Fitting a Smooth Intensity Surface* Computer-Aided Design, Vol. 27, No. 8, pp. 595-603, 1995.

[Kame03] M. Kameyama, Y. Kato, H. Fujimoto, H. Negishi, Y.Kodama, Y. Inoue, H. Kawa, *3D Graphics LSI Core for Mobile Phone Z3D,* Eurographics/siggraph Graphics Hardware 2003.

[Kirk90] D. Kirk, D. Voorhies, *The Rendering Architecture of the DN10000VS* Computer Graphics vol. 24, pp. 299-307, August 1990.

[Kirk92] D. Kirk, O. Lathrop, D. Voorhies, *Quadratic Interpolation for Shaded Image Generation* Patent Nr: US5109481, 1992.

[Lee01] Y. C. Lee, C. W. Jen, *Improved Quadratic NormalVector Interpolation for Realistic Shading* The Visual Computer, 17, pp. 337-352, 2001.

[Phon75] B. T. Phong, *Illumination for Computer Generated Pictures* Communications of the ACM, Vol. 18, No 6, June 1975.

[Powe77] M. J. D. Powell, M. A. Sabin, *Piecewise Quadratic Approximations on Triangles* ACM Trans. Math. Software Vol. 3, pp 316-325, Dec. 1977.

[Saxe96] E. Saxe, A. A. Lastra, M. Hughes, *Higher-Order color Interpolation for Real-Time Radiosity Displa,y* UNCCH Dep. of Computer Science Technical Report TR96-023, 1996.

[Seil98] L. Seiler, *Quadratic Interpolation for Near-Phong Quality Shading* Proceedings of SIGGRAPH 98: conference abstracts and applications, Page 268, 1998.

[Wynn01] C. Wynn, *Implementing Bump-Mapping using Register Combiners,* Newton-Raphson fast combiner normalization technique. http://developer.nvidia.com, 2001.

Hierarchical texture compression

Jerzy Stachera
Institute of Computer Science
Warsaw University of Technology
Ul. Nowowiejska 15/19
00-665 Warszawa, POLAND
J.Stachera@ii.pw.edu.pl

Przemyslaw Rokita
Institute of Computer Science
Warsaw University of Technology
Ul. Nowowiejska 15/19
00-665 Warszawa, POLAND
P.Rokita@ii.pw.edu.pl

ABSTRACT

Texture mapping is a technique for adding visual realism to the computer generated images. As the level of realism increases with the number and the resolution of textures, we are faced with the problem of limited texture memory space. Moreover, in order to alleviate the aliasing artefacts many graphics systems use the mip-mapping technique which needs to store additionally the texture pyramid. We propose an algorithm for texture compression which is characterized by low computational complexity, random access to compressed data and the hierarchical texture representation. The proposed hierarchical texture compression algorithm (HiTC) is based on a block-wise approach, where each block is subject to the modified fractal compression method and is partly represented by Laplacian pyramid. This allows us to incorporate the mip-map structure into the compressed texture and perfectly suits for real-time computer graphics applications.

Keywords

Texture compression, fractal compression, Laplacian pyramid, mip-mapping.

1. INTRODUCTION

The computer generated scenery composed of wire-frame models was the first step into the virtual world. The models could only reflect the 3D structure of objects and were devoid of any other form of visual information. That was changed by the introduction of texture mapping techniques. Textures in their basic form are images which are applied upon 3D wire-frame models. When mapped onto an object surface, the color of the object surface is modified by the corresponding color from the texture. In general, the process of texture mapping takes several steps. Since textures are represented by an array of discrete samples, a continuous image must first be reconstructed from these samples. In the next step, the image must be warped to match any distortion and filtered to remove high-frequency aliasing artefacts. In most cases, the filtering is done by mip-mapping [Wil83], which introduces additional memory cost in the form of texture pyramid.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

Current applications of textures are much wider and textures are used to control most of the visual parameters of the object surfaces such as transparency, reflectivity, bumpiness, roughness. This greatly increases the realism of the virtual objects and pushes the hardware resources to the limits. To simulate the real world at the interactive frame rate, it is necessary to have fast and random access to a large number of high resolution detailed textures. Thus, the bandwidth and storage requirement for textures introduces the need to use more efficient texture compression methods.

2. PROBLEM DEFINITION

Although a texture can be regarded as a digital image, most of classical image compression methods cannot be applied to textures. There is a strong need for efficient, highly specialized texture compression algorithms.

According to Beers [Bee96] and our findings when designing a texture compression algorithm the following aspects must be taken into consideration:

Decoding speed. It is the essential feature which allows for rendering directly from the compressed textures. As texture compression is mostly used in real time computer graphics the decompression

algorithm must be designed to allow high frame rates.

Random access. Since the mapping process introduces discontinuous texture access in the texture space, it is difficult to know in advance how the texture will be accessed. Thus, the methods which may be considered for fast random access are based on fixed length codes.

Hierarchical representation. To deal efficiently with level of detail and mip-map texture representations this requirement must be fulfilled. Hierarchical structure allows for rendering directly from the compressed texture represented on number of different resolution levels.

Compression rate and visual quality. The difference between textures and images is that the images are viewed on their own and they are presented in the static content, while the textures are the part of the scene which usually changes dynamically. Thus, the loss of information in texture compression is more acceptable and the compression ratio is more important issue.

Encoding speed. Texture compression is an asymmetric process, in which the decompression speed is crucial and the encoding speed is useful but not essential.

3. PREVIOUS WORK

All existing texture compression algorithms can be divided into three major groups: block truncation coding and local palettes, vector quantization, transform coding.

3.1 Block truncation coding and local palettes.

The block truncation coding (BTC) was introduced for image compression by Delph and Mitchel [Del79]. The method represents image 4×4 block by two 8-bit gray scale values and index map. Each pixel in the block references one of the gray scale values by 2 bit index from the index map. This corresponds to 2 bits per pixel (bpp) compression. Although, the primary application of BTC was not the texture compression, many other proposed texture compression methods are based on it.

An extension of BTC to represent color images, called color cell compression (CCC) was proposed by Campbell [Cam86]. The two 8-bit values were used for storing the index into a color palette. That representation allowed compressing the color images to 2bpp. Even though, the additional cost of storing a color palette for every texture and indirect data access requiring two memory transactions were prohibitive for some real time application, it was suggested by the Knittel to implement it in the texturing hardware [Kni96]. In terms of image quality both BTC and CCC are characterized by the block effect. This effect is a result of two contrary factors, namely independent block compression and limited number of colors used for the block representation.

The further extension of those two described methods was the S3TC texture compression method introduced by Iourcha [Iou99]. The S3TC represents a 4×4 block by four 16 bits (RGB565) color values and the index map with 2 bits per index. Two base colors are stored explicitly in the compressed block and the others are linearly interpolated from those two during the decompression process. Thus, the final size of the block is equal to 64 bits which gives 4bpp. The FXT1 texture compression method developed by 3DFX added a few more modes to S3TC [3DF99]. The first mode is the same as S3TC and the other compress 4×8 blocks with local palette with four and eight colors interpolated depending on the mode from two or three base colors. Another S3TC like method was a part of POOMA texturing system for low-cost devices [Ake03]. It represented a 3×2 block by two base colors and used block overlapping instead of a texture caching to reduce the memory access.

The texture quality of S3TC based methods is significantly better than CCC and BTC, but it still suffers from the block effect partly introduced by linear interpolation of colors. A number of solutions were proposed to tackle that problem. The color distribution approach proposed by Ivanov and Kuzmin allowed to share colors between neighbour blocks [IVA00], thus representing a block by a larger number of unique colors. Levkovich-Maslyuk et al. allow colors to be chosen from an RGB tetrahedron, and partitioned the block into sub-pallets for better approximation of block original color distribution [Lev00].

Completely different approach was taken by Fenney [Fen03]. On the basis of the fact that low-pass filtered signals are often a good approximations of original signal, his method used two low resolution images and full-resolution low precision modulation signal to represent a texture. Another approach was proposed by Ström and Akenine-Möller in PACKMAN system and its extension iPACKMAN [Str04][Str05]. Proposed methods represent a block by a single base color and an index map with indices into a codebook which values modulate the pixel luminance.

Although texture compression methods based on BTC are currently the mainstream in computer graphics hardware, they do not address the problem

of hierarchical representation. Thus, the main features that characterize them, namely - decoding speed and fast random access, may come at great expense when applied to level of detail and mip-map texture representations.

3.2 Vector quantization.

Beers et al. proposed a method for texture compression based on vector quantization (VQ) with mip-map texture representations [Bee96]. The first mip-map (original texture) was represented by 4×4 blocks from the codebook. On the basis of the first mip-map codebook the second and the third were created by averaging each 4×4 codeword respectively to 2×2 and 1×1 codewords. The final extended codebook was created by concatenating the 4×4 codeword with the corresponding sub-sampled 2×2 and 1×1 codewords. This representation achieved significant compression of 1bpp at the cost of lower reconstruction quality due to codeword sub-sampling.

Method proposed by Kwon and Chong used Interpolative Vector Quantization to represent a texture pyramid used for mip-mapping [Kwo00]. This allowed reducing correlation between mip-map levels. Additionally, the method needed to store two codebooks corresponding to low and high frequency texture terms and to interpolate the low frequency part of the block during the decompression.

Tang and Zhang in their texture compression method address the problem of texture regions visual importance [Tan05]. The codebook in this method is constructed taking into account such issues as visual importance and texture mapping distortion.

Generally, the VQ based methods suffer from two major problems: indirect data access and codebook handling. To retrieve single texture element we need two memory accesses. The codebook size can be too expensive for implementation in hardware and impose additional cost on texture caching when used with mip-maping.

3.3 Transform coding.

The most common transform method used in image compression which was applied to textures was Discrete Cosine Transformation (DCT). Talisman texturing systems (TREC) solved the problem of variable length coding of DCT by preserving the DC components without DPCM [Mic97]. Moreover, TREC used index table and link list to address each block, resulting in block random access. More elaborated texture compression scheme based on DCT was proposed by Chen and Lee [Che02]. They use adaptive quantization of 8×8 blocks, by assigning quantizer scale factor to each block. Thus, each block could be encoded to fixed-length code. Although those methods resulted in compression ratio higher than BTC methods and comparable to VQ methods with better image quality, they were too expensive for hardware implementation due to large block size, high complexity of inverse transform and lack of random access on texture pixel level.

Following the DCT more attention in TC is gaining the discrete wavelet transform (DWT). This is the result of the multi-resolution representation. Pereberin introduced the compression method based on Haar wavelet to encode three adjacent levels of mip-map pyramid [Per99]. Mapping the texture to YUV color space and reducing the insignificant coefficients allowed him to achieve average 4bpp compression. Representing a texture as a Wavelet Coefficient Tree (WCT) in the form of coefficient texture and the index texture was proposed by Candusi and DiVerdi [Can05]. Though, the DWT poses multi-dimensional feature, which makes it superior for hierarchical texture representation, it is not obvious how to reduce the insignificant coefficients to obtain the random access without severely reducing the compression ratio.

4. HIERARCHICAL TEXTURE COMPRESSION.

As the level of detail representation and filtering based on mip-mapping technique is ubiquitous in real-time applications, it is needed for textures to have a form of hierarchical structure.

It can be seen in image processing field that hierarchical approach gives an effective solution for compressing images. A good illustration being wavelet decomposition [Tau02] or Laplacian pyramid [Bur83]. These methods store the hierarchy explicitly compared to other class of methods based on fractal theory. Fractals are characterized by super-resolution property and while they can be represented on different resolutions levels, the hierarchical structure is not explicitly stored.

Even though, the hierarchical and multi-resolution feature of wavelets and fractals seems promising for texture compression applications, they are generally unsuitable. On one hand, wavelets methods need tree-walk procedures which require multiple accesses to memory. On the other hand, fractals during decompression need information from different parts of the texture. Moreover, they both are characterized by variable length code which makes them inferior to widely used block truncation coding methods.

Taking into consideration the requirements of texture compression we propose a method (HiTC) that combines the hierarchical representation with

block-wise approach. This allows us to join the advantages of both approaches.

5. THE ALGORITHM DESCRIPTION.

The proposed algorithm is based on fractal and wavelet theory. Since the final structure of algorithm was subject to number of simplifications, the next part of the paragraph will explain step by step the process that was carried to derive it.

5.1 Introduction.

One of the most important factors in BTC texture compression methods is the size of the compressed texture block, which is in most cases equal to 4×4. It is the result of a balance between the image quality and compression ratio. Moreover, in texture compression applications it is considered as optimal for hardware implementation. But if we take into account the Fractal compression methods, the block that is subject to compression constitutes the whole image. The fact that the base fractal scheme does not address the problem of local compression [Fis95] makes the process of decompression computationally expensive. The work on the local fractal transform was carried on by the author resulting in the local fractal compression algorithm [Sta05]. That algorithm is based on quadtree structure. It allows for local decoding and random access on block level. The block size is restricted by the image quality and is not lower than 32×32. Although the local fractal compression algorithm achieves high compression ratios and close to real-time decompression, the quadtree structure makes it difficult for hardware implementation and it does not allow for random access on texture pixel level.

In our new method (HiTC) we take advantage of the local fractal compression algorithm for 4×4 blocks compressed independently. To overcome the problem of quadtree structure we use regular partitioning of compressed block to 2×2 range blocks R. The local domain pool for compressed block consists of only one domain block being the whole compressed block $D=B$. The resulting fractal code is a subject to further compression using Haar wavelet or Laplacian Pyramid.

5.2 Compression.

The process of texture compression in our algorithm (HiTC) consists of the following steps:

1. The texture is partitioned into 4×4 blocks.
2. Each block is subject to the modified local fractal compression.
3. The fractal code of each block is further compressed by Haar wavelet decomposition or is represented by Laplacian pyramid.

If we consider the texture block B, the modified local fractal compression is derived from fractal block-based method [Fis95] [Woh99] and is done by:

1. The 4×4 block B is divided into four non overlapping range blocks (4 squares of size 2×2).
$$\{R_j\},\ j=1,2,3,4$$
2. Each range block R is matched to the domain block D (in this case $D=B$) by computing the optimal coefficients of the transformation:
$$\hat{R}=s\cdot\varphi(D)+o \tag{1}$$
which minimize the error $d(\hat{R},R)$ in this case the root mean square error:
$$rms=\sum_{i=1}^{4}\left(s\cdot\varphi(d_i)+o-r_i\right)^2 \qquad r_i\in R_j,\ d_i\in D \tag{2}$$

where, $\varphi(\cdot)$ is the spatial contraction function which averages four adjacent domain block elements and then maps the averaged values onto range block applying one of eight isometries (sym). Next, the resulting range block values are scaled by s (scaling) and added to o (offset) coefficient [Fis95] [Nin97].

Thus, after compression each range block R_j is represented by a triple $\{s_j,o_j,sym_j\}$.

Pi et al. in the context of image coding proposed to replace the offset coefficient o (equation 1) by range block mean $\bar{r}$ which led to transformation of the form [Pi03]:
$$\hat{R}=s\cdot\varphi\left(D-\bar{d}\right)+\bar{r} \tag{3}$$

where $\bar{d}$ is the average domain value, $\bar{r}$ represents range block DC component and s (scaling) in this context is related to range block AC component.

They proved that the range block mean values are good approximation of image DC component. Consequently, starting from the initial image equal to the range-average image (DC image) could lead to faster convergence. Moreover, further iteration on DC image change only AC component, thus reducing the iterations on average to two.

Even though, the convergence of the algorithm proposed by Pi is faster we still need to compute the domain block average at each iteration. In our method, we solve this problem by making the domain block average as one of the compressed block coefficients. This is only cost-effective due to block size restriction to 4×4.

Finally, we obtain the following compressed block structure which is represented by four range

blocks coefficients $R_j = \{s_j, \bar{r}_j, sym_j\}, \quad j = 1,2,3,4$ and domain block average $\bar{d}$. As can be seen from the figure 1 two levels of mip-map are the part of our fractal code. Thus, we can further apply Haar transform or Laplacian pyramid to increase the compression ratio.

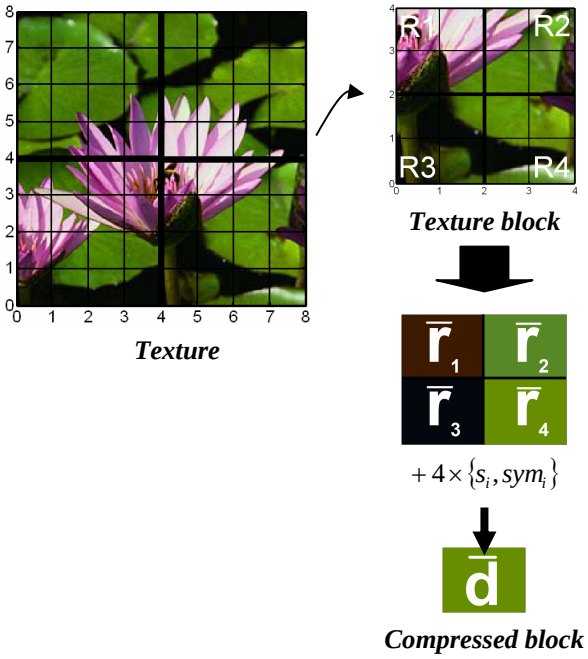


Figure 1. Hierarchical texture compression (HiTC).

A simple example reveals the effectiveness of our method. For Lenna image if we store only the mip-map pyramid and set all the rest coefficients to default values ($s_j = 0.75, sym_j = 0$, one to one mapping) we achieve $psnr = 30{,}65dB$, $rmse = 7{,}47$ and $C_R = 4.2:1$. In this case, we obtained compression ratio of BTC methods at the same time storing explicitly two levels of mip-map pyramid. Moreover, the same hierarchical structure represented by wavelets could not be compressed without wavelet coefficient quantization. Thus, each memory access for wavelets require a de-quantization step [Per99], while in our method it is done directly.

5.3 Coefficients allocation.

The next problem that must be solved is the coefficients allocation. Namely, given the compressed texture block representation we search for optimal coefficient bit allocation with respect to compression ratio and image quality. Therefore, each of the coefficients was independently subject to uniform and non-uniform quantization with default coefficients values set to: scaling 7-bits, offset 5-bits and symmetry 3-bits [Fis95].

5.3.1 Scaling.

The scaling coefficient was subject to uniform quantization in the range $[-1.5,1.5]$. The value $s_{\max} = 1.5$ was chosen experimentally on the basis of the reconstruction error. The results of scaling quantization revealed that choosing the quantization levels with precision higher than 1/32 has little impact on image quality as it was indicated by Ning Lu [Nin97] (fig. 2). Moreover, restricting the scaling values to positive resulted in $1dB$ lower reconstruction error, which may be acceptable in texture compression. Applying the non-uniform quantization showed little improvement. The most noticeable difference was for one quantization level where the reconstruction error was reduced by $0.5dB$. It corresponded to reconstruction values $s \approx \{0.26, 0.86\}$, as opposed to $s \approx \{0, 0.75\}$ for uniform quantization. The compression was also checked for constant scaling coefficient. The optimal value was equal to $s = 0.5$. But since the constant value introduce the artefacts to high frequency term of the image, it seems reasonable to use at least one level of quantization for scaling coefficient.

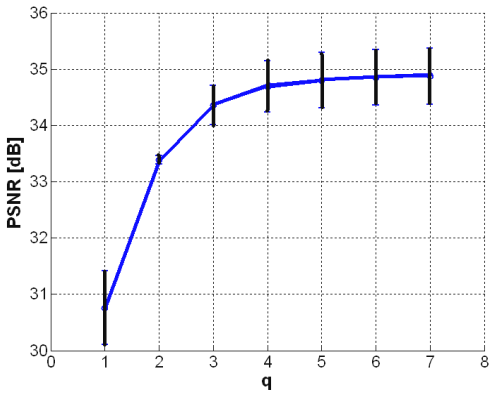


Figure 2. Scaling coefficient quantization for Lenna, with error bars indicating the difference between uniform and non-uniform quantization.

5.3.2 Symmetry.

The symmetry coefficient is responsible for mapping the domain block onto the range block [Fis95] (there are eight ways to map a square onto another). As it can be seen from the distribution of symmetry coefficient for positive scaling coefficient, the trivial mapping, which maps averaged domain values on the same positions in the range block is dominant. Additionally, which is not showed here for positive and negative scaling coefficient the second dominant mapping is the mapping which rotates the block by 180^0. Thus, it can be suggested to restrict the compression process to use only this trivial mapping and positive scaling coefficient. Moreover, if we remove the trivial mapping from the

distribution diagram, the rest is uniformly distributed and therefore choosing any particular mapping gives no improvement. The only visual difference can be observed for at least four symmetries (fig. 4, q2 and q3)

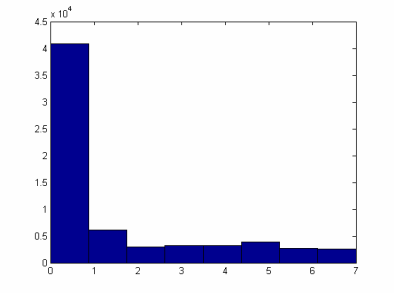


Figure 3. Symmetry coefficient distribution (0 – trivial mapping).

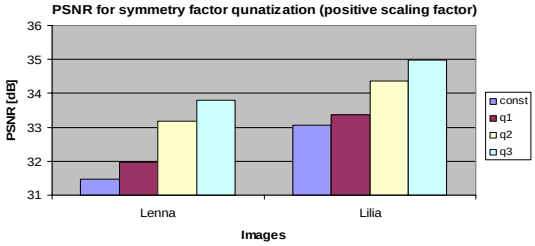


Figure 4. Symmetry coefficient quantization.

5.3.3 Mip-map.

The result of the texture compression process is the set of parameters, where the values $\{\bar{r}_j\}$, $j=1,2,3,4$ and $\bar{d}$ represent the second and the third (the lowest) level of the mip-map pyramid. Therefore, the next step is to apply compression to those parameters. We considered two approaches: the Haar decomposition [Per99] and Laplacian pyramid representation [Bur83]. We chose the Laplacian pyramid, since it fits more closely to decompression process and at the same time is less expensive computationally. It can be seen from equation 3, we can reduce the computation of domain difference $D-\bar{d}$ since it is a part of Laplacian pyramid representation. We represent the mip-map pyramid by storing the lowest level explicitly and the second level as the difference terms $d_j=\bar{r}_j-\bar{d}$, $i=1,2,3,4$. Moreover, we take advantage of color space with separated luminance and chrominance to coarsely approximate the chrominance data.

5.4 Block structure and decompression.

Generally, the structure of compressed texture block is represented by the triples $\theta=\{s,\bar{r},sym\}$, where each corresponds to one of four range blocks. Taking into account the previous paragraph we can reduce this representation. The proposed bit allocation scheme is presented below.

The scaling coefficient can be represented by one bit and in the case of the uniform quantization the reconstruction values are $s\approx\{0,0.75\}$. Since when using only one quantization level the value of the coefficient is on one third equal to zero, thus the process of decompression is usually reduced to retrieving the block mean value (equation 3).

The symmetry coefficient is dominated by one to one mapping, thus it can be safely removed from the block structure.

The mip-map representation in our case is compressed by using the Laplacian pyramid representation. But before that we take advantage of color space conversion to reduce the data. We chose the YC_bC_r color space to compress the chrominance data. We allocate 7-bits for luminance difference terms and 3-bits for chrominance difference terms.

The final block bit allocation structure for the color components consists of:

- block average value $\bar{d}$ (8-bits),
- difference terms d_j, $j=1,2,3,4$ (7-bits – luminance and 3-bits chrominance),
- scaling coefficient s_j, $j=1,2,3,4$ (1-bit).

This allocation scheme gives 88-bits (40-bits for luminance and 2x24-bits for chrominance components) and compression ratio $C_R=5.72:1$ (the compression of S3TC for the same set of mip-maps would be $C_R=4.57:1$ [Per99]).

The decompression process for color component (Y, C_b or C_r) simulating three level mip-map pyramid can be described in the following steps:

```
ColorComp getTexel(int x, int y, int mipLevel) {
     CBlock_x = x/4; CBlock_y = y/4;
     If (mipLevel >= 2) {
          getCoeff(CBlock_x, CBlock_y, &d);
          return d;
     }
     else if (mipLevel == 1) {
          getCoeff(CBlock_x,CBlock_y,&d,&d1);
          return d + d1;
     }
     //mip level 0
     getCoeff(CBlock_x,CBlock_y,&d,&s,&d1,&d2);
     if (s == 0)
          return  d + d1;         //r = d + d1
     else
          return s*(d2) + d + d1; //s*(di-d)+r;
}
//getCoeff() – unpacks the data from the block
```

6. RESULTS.

The presented method (HiTC) was compared with S3TC algorithm on a sequence of test images (fig. 5) and the rendered OpenGL scene (fig. 6). The error was measured by computing the peak signal to

noise ratio on luminance component. The luminance component was chosen for the reason of its visual importance.

The HiTC shows better reconstruction in the regions of smooth color variance as opposed to the S3TC. The S3TC reconstruction error is uniformly distributed over whole image which can be easily seen in the form of the block effect. This effect is especially magnified on the block edges where the colors do not lie on a line in a RGB color space. The same effect in minor form can be observed for HiTC. It is reduced in our method by approximating the colors of the texture by the mip-map pyramid.

The final reconstruction errors for the textures should be taken with caution. Since they are part of the visualized scene, they are subject to mapping and filtering process. Although, the compression of single texture gives some differences when comparing with uncompressed texture, the final rendered scene is visually indistinguishable. Both methods achieved the peak signal to noise ratio for rendered scene higher than $40dB$.

The problem of decompression cannot be fully addressed without hardware implementation. However, since our method addresses the problem of storing three levels of mip-map in one block, we could expect the lowest complexity of accessing the texture pixel comparing to other methods.

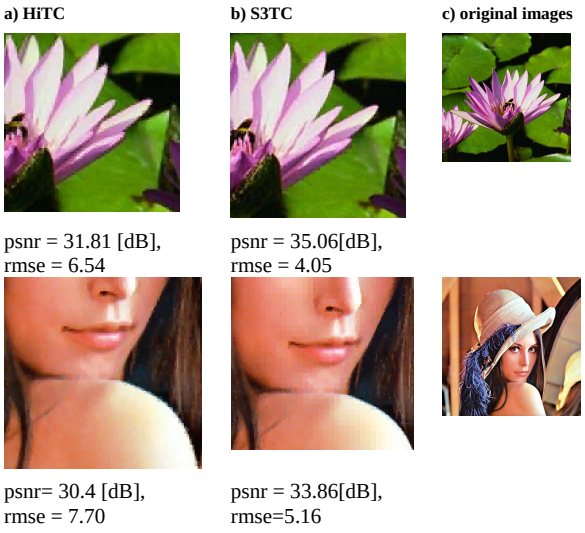


Figure 5. Texture reconstruction error.

7. CONCLUSION.

We have proposed a new approach for texture compression. The major advantage of our fractal block-based approach is a hierarchical representation, which allows for:

- direct decompression, which does not need any iteration since all the coefficients are stored explicitly in the compressed block structure,
- texture hierarchical representation, which is the result of our modified local fractal compression process,
- low computational complexity - to compute first level we need to perform only one multiplication and one addition, the second level needs only one addition and the third level is stored explicitly.

HiTC has the advantage on currently used BTC methods that the process of mip-mapping is addressed by the compressed texture block (table 1). This can be seen when used with trilinear filtering. Our method outperforms currently proposed solutions, since it can access three levels of mip-map directly. The hardware implementation is simple and does not require any external structures which for example needs vector quantization methods. There is no pre-processing step related to coefficient de-quantization which is common for wavelet compression. The computational complexity was reduced to minimum with the aim of real-time application. Moreover, all the requirements on texture compression method are fulfilled (table 1), thus making it superior for high performance rendering architectures.

8. REFERENCES

[3df99] 3dfx. FXT1: White paper. 3dfx Interactive. http://wwwdev.3dfx.com/fxt1/fxt1whitepaper.pdf, 1999.

[Ake03] Akenine-Möller T., Ström J. Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. ACM Transactions on Graphics, 22, 3 (2003), 801–808.

[Bee96] Beers A. C., Agrawala M., Chaddha N. Rendering from compressed textures. Siggraph 1996, pp. 373–378, July 1996.

[Bur83] Burt P.J. and Adelson E.H. The Laplacian pyramid as a compact image code. IEEE Transactions on Communications, 31:532–540, 1983.

[Cam86] Campbell G., Defanti T. A., Frederiksen J., Joyce S. A., Leske L. A., Lindberg J. A., Sandin D. J. Two Bit/Pixel Full Color Encoding. In Proceedings Of Siggraph (1986), Vol. 22, Pp. 215–223.

[Can05] Candussi N., DiVerdi S., Hollerer T. Real-time Rendering with Wavelet-Compressed Multi-Dimensional Textures on the GPU. Computer Science Technical Report 2005-05, University of California, Santa Barbara.

[Che02] Chen C.-H., Lee C.-Y. A JPEG-like texture compression with adaptive quantization for 3D graphics application. The Visual Computer, vol. 18, 2002, pp. 29-40.

[Fen03] Fenney S. Texture Compression using Low-Frequency Signal Modulation. In Graphics Hardware (2003), ACM Press, pp. 84–91.
[Fis95] Fisher Y. (Ed.). Fractal Image Compression: Theory and Application to Digital Images. Springer Verlag, New York, 1995.
[Iou99] Iourcha K., Nayak K., Hong Z. System and Method for Fixed-Rate Block-based Image Compression with Inferred Pixels Values. In US Patent 5,956,431 (1999).
[IVA00] Ivanov D., Kuzmin Y. Color Distribution – A New Approach to Texture Compression. In Proceedings of Eurographics (2000), vol. 19, pp. C283–C289.
[Kni96] Knittel G., Schilling A., Kugler A., Strasser W. Hardware for Superior Texture Performance. Computers & Graphics 20, 4 (July 1996), 475– 481.
[Kwo00] Kwon Young-Su, Park In-Cheol, and Kyung Chong-Min. Pyramid Texture Compression and Decompression Using Interpolative Vector Quantization. Proceedings of 2000 International Conference on Image Processing, vol. 2, pp.191-194, Sep. 10-13, 2000.
[Lev00] Levkovich-Maslyuk L., Kalyuzhny P. G., and Zhirkov A. Texture Compression with Adaptive Block Partitions. ACM Multimedia 2000, Nov.2000.
[Mic97] Microsoft, "Escalante hardware overview. Talisman". Graph Multimedia Syst, pp 89–106.
[Nin97] Ning Lu, Fractal Imaging, Academic Press, 1997.
[Per99] Pereberin A.V. Hierarchical Approach for Texture Compression. Proceedings of GraphiCon '99, 1999,195–199.
[Pi03] Pi M., Basu A., and Mandal M. A new decoding algorithm based on range block mean and contrast scaling. IEEE International Conference on Image Processing (ICIP), vol. 2, pp. 241-274, Barcelona, Spain, September 14-17, 2003.
[Sta05] Stachera J., Nikiel S. Large textures storage using fractal image compression. to be published in Computational Imaging And Vision book series, Kluwer 2005.
[Str04] Ström J., Akenine-Möller T. PACKMAN: Texture Compression for Mobile Phones. In Sketches program at SIGGRAPH (2004).
[Str05] Ström J. and Akenine-Möller T., iPACKMAN: High-Quality, Low-Complexity Texture Compression for Mobile Phones, Graphics Hardware 2005, pp. 63-70, 2005.
[Tan05] Tang Y., Zhang H., Wang Q., Bao H. Importance-Driven Texture Encoding Based on Samples. Computer Graphics International 2005.
[Tau02] Taubman D., Marcellin M. W. JPEG2000: Image Compression Fundamentals, Standards and Practice. Kluwer, Boston, 2002.
[Wil83] Williams L. Pyramidal Parametrics. Computer Graphics (SIGGRAPH'83 Proceedings), Pages 1-11, July, 1983.
[Woh99] Wohlberg B., Jager G.,"A Review of the Fractal Image Coding Literature",IEEE Transactions On Image Processing,Vol. 8, No. 12, December 1999

Method	Random Access	Simple decoding	Simple hardware implementation	Hierarchical representation	Compression Ratio	Image quality
BTC	Yes	Yes	Yes	No	Average	Average
VQ	Yes	Yes	No	No	High	Average
DCT	No	No	No	No	High	High
DWT	No	No	No	Yes	Highest	Highest
Fractal	No	Yes	No	Yes	Highest	High
HiTC	**Yes**	**Yes**	**Yes**	**Yes**	**Average**	**Average**

Table 1. Texture compression methods comparison

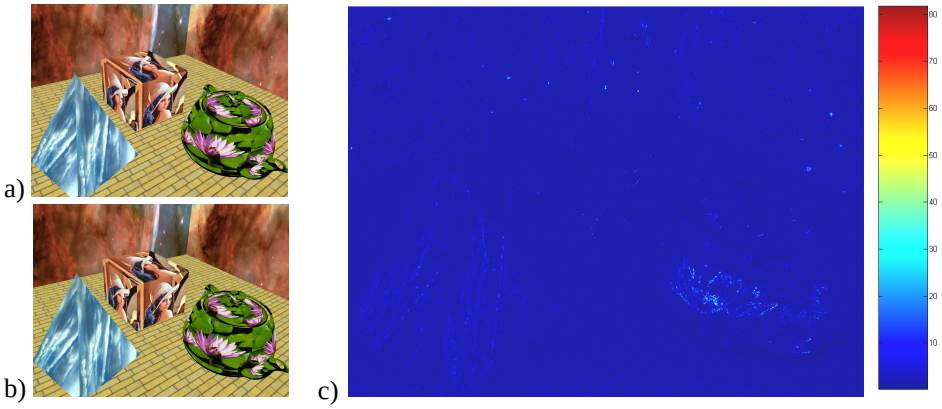


Figure 6. Scene rendered in OpenGL. a) normal view with uncompressed textures and with c) HiTC compressed textures. c) Scene error image for HiTC textures (psnr = 43dB, rmse = 1.82) .

Enhanced Billboards for Model Simplification

Phongvarin Vichitvejpaisal
Faculty of Computer Engineering
Chulalongkorn University
Pathumwan
Bangkok 10330, Thailand
matt_varin@yahoo.com

Pizzanu Kanongchaiyos
Faculty of Computer Engineering
Chulalongkorn University
Pathumwan
Bangkok 10330, Thailand
pizzanu@cp.eng.chula.ac.th

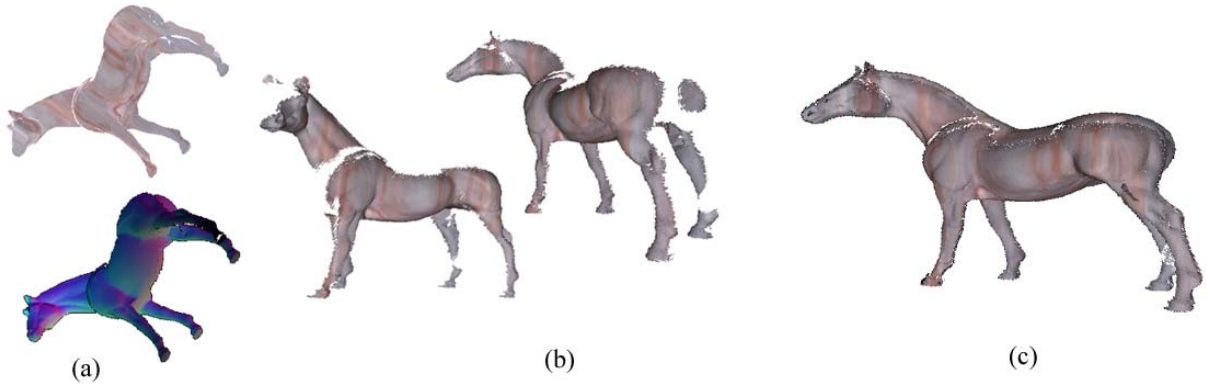


Figure 1: (a) An enhanced billboard consisting of a color map, a transparency map, a depth map and a normal map encoded into two images. (b) A horse model rendered with an enhanced billboard shown in (a), therefore representing only some portion of the model. (c) Complete horse model rendered with all enhanced billboards.

ABSTRACT

A set of billboards can represent 3D models for extreme simplification in real-time rendering. Unlike conventional polygon method, billboard-based technique has the rendering time of the model proportional to its contribution to the image. Thus, it has an automatically built-in Level of Detail. However, previous techniques still have limited viewing angle and do not accurately represent the model. We present an adaptive rendering method using enhanced billboards. First, each enhanced billboard, representing portion of the model, is defined to have four maps: depth map, normal map, color map and transparency map. The model is then projected onto a number of viewing planes in different viewing directions. Consequently, these enhanced billboards are rendered based on ray-height-field intersection algorithm implemented on GPU. This representation can maintain the geometry and the silhouette of the model with no limit in viewing direction. Moreover, real-time rendering is supported.

Keywords

image-based rendering, displacement, billboards, real-time rendering, simplification

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

1. INTRODUCTION

One of the important problems in computer graphics is how to handle the overwhelming complexity of the screen. A detailed scene can give a more attractive and realistic look but lower the performance. A typical scene nowadays may contain up to millions of polygons. In addition, programs such as games are performance-critical applications. Many techniques have been invented to manage the tradeoff between speed and quality of the picture. These techniques deal with new model representations as well as new rendering algorithms. The billboard is one of the various image-based model simplification methods

with advantages in its simplicity and rendering speed. Typically, a billboard represents a 3D model with an image. The image is placed at the position of the model and rendered as a texture mapped polygon quad. The image-based property of the billboard enables its rendering time to scale linearly with its screen contribution. Unfortunately, the billboard has a lot of limitations. For example, it must be viewed with narrow viewing directions, otherwise the 2D structure of the image will be noticeable.

This paper shows how the billboard can be enhanced to produce high quality images without such limitations. We propose a new representation for the model, which is a series of images. Each image consists of a color map, a transparency map, a normal map and a depth map, or displacement map, called *enhanced billboards*. The enhanced billboard image is projected from different direction of the model. Therefore, we can represent all the information of the 3D model. The enhanced billboards are shown in Figure 1. Previous simplification techniques typically store only some information of the source model, and, therefore, much of the information is lost. We use a new ray-height-field intersection algorithm to render the enhanced billboards. Our method emphasizes on the proportion of processing time compared to the contribution of the outcome image. Thus, the enhanced billboard is effective in rendering rich detail models in distant scenes with unnoticeable artifacts. It maintains the silhouettes, parallax effect and smooth surface without any cracks.

2. RELATED WORK

Many image-based approaches are used to simplify the model. The Impostor Technique replaces distant geometry by projecting an object or a portion of the screen onto a plane and rendering it as an image. Bump Mapping [Bli78; Per97] uses the normal map to perturb the normal of the model, thus increasing the detail of the model. However, it cannot render the silhouette and shadow. Displacement Mapping [Hir04; Pha96; Sch99; Smi00], in contrast, simulates the actual geometry of the surface and can produce the silhouette and shadow effect. Nevertheless, displacement mapping increases the amount of micro-polygons to capture the detail of the model, which is not appropriate for real-time application.

The layered depth image [Sha98] is developed to handle the parallax effect. It is an image that contains depth information. The drawback of the layered depth image is that it can only be viewed at a narrow range.

Compared to our enhanced billboard, which uses a series of image from different views of the model, our method does not suffer from limited viewing directions. Meyer et al. [Mey98; Mey01; Phi04; Jak00] proposed a method to render a forest scene. They used a group of billboards, each being the sliced image of some group of trees. This method faced a problem with the viewing region, which makes it more suitable for a bird eye's view of the forest. The Billboard Cloud Technique [Xav03] also uses a number of billboards, each representing some portion of the model. The billboard cloud is rendered as a conventional texture map polygon. Because it uses a plane to approximate the nearby geometry, the billboard cloud results in cracks in the image.

Another simplification technique is Mesh Decimations [Bax02; Gar97; Hop04; Jes02], which reduces the number of polygons rendered on the screen. The number of polygons is decreased by using edge-collapse or vertex-removal algorithm. Thus, the quality of the model is lower than the original, and the results may be rough models with discontinuity. Some of the key visual details such as the nose of the face can be lost. These techniques depend on the property of the input model, such as topology or connectivity. Thus, a different algorithm has to be used for a different type of model. Another drawback is that this method requires that the model be stored in various resolutions. This will increase the storage cost and introduce a pop-up artifact when changing to different resolutions of the model. The Progressive Mesh [Hop96; Hop97] Technique is invented to solve the pop-up problem. It has an auxiliary data structure to smoothly add and collapse the vertex of the object. Thus, it can continuously increase or decrease the quality of the model without any noticeable effects.

Wang et al. [Wan03; Xi04] proposed a five dimensional representation GDM (general displacement map) to quickly render the displacement model. This technique stores the pre-compute distances from each displaced point to a reference surface. It requires a lot of storage and needs to be compressed.

Relief Texture [Fab05; Oli00] Mapping bears similarities to our work. The relief texture is mapped onto the surface to capture the detail of the model. It has a normal map and a depth map like ours. The relief texture map can capture the parallax effect correctly, but the silhouette effect cannot be produced. In contrast, our method is designed to render the complete model with silhouette information, not just the detailed surfaces.

Figure 2: The first five enhanced billboards of the horse model encoded as color-alpha images and normal-depth images.

3. ENHANCED BILLBOARDS

We introduce *enhanced billboards* as an alternative for representing an arbitrary polygon model. An enhanced billboard consists of a color map, a transparency map, a normal map and a depth map. These maps are stored in two images. One image stores color map in the RGB channel and the transparency map in the alpha channel. The other image stores the normal map in the RGB channel and the depth map in the alpha channel since an image is a 2D data structure. The depth map is used to store the third dimension of the model encoded as depth information. The normal map is used in the shading algorithm, similar to a bump map.

The model is projected onto a set of planes. Each plane captures some polygons in the model. The projected polygon stored as an enhanced billboard includes all the color, normal and depth information of that polygon. The direction of the projected plane is chosen to minimize the number of planes needed to capture all the polygons in the model. With a smaller number of enhanced billboards, less storage and time will be required for rendering. Thus, we use the greedy algorithm that relies on the normal vector of the polygon to choose the projection plane. The enhanced billboards are rendered with our new ray-height-field algorithm by the help of GPU.

The enhanced billboards can store all the information of the model, so no information is lost nor approximated, thus enabling enhanced billboards to be viewed in any direction with the correct silhouette. The amount of time used to render the enhanced billboards scales linearly to the number of pixels they cover. Thus, enhanced billboards are suitable for representing distant objects while preserving the geometric feature of the model.

3.1 Building Enhanced Billboards

This section shows how the polygon model is represented as enhanced billboards (see Fig 2.). An enhanced billboard can only represent polygons that are facing it. In addition, an enhanced billboard cannot store the polygons that occlude one another in the same image. Thus, the polygons that are occluded by some other polygons have to be stored in another enhanced billboard. We have to choose a robust way to pick projected directions that will result in small number of enhanced billboards used, and the chosen technique is based on using the normal vectors of the polygons.

First, we build an array of direction buffers. Each element of the array represents a direction in space. We use an array size of 36x18 to represent all the directions around the object. Each array element is used to accumulate the number of polygons that is facing the direction of the array element. Then, we choose the direction that has the most number of polygons. Although our method may not guarantee the smallest number of billboards, our greedy algorithm is easy to implement and still results in a small number of billboards.

We then have the direction for projecting the model. However, we cannot project the model yet because the polygons may be overlapping or the polygon may be far apart from each other. Our enhanced billboards will not store overlapping polygons in the same image. Moreover, enhanced billboards must have the same limited depth value. This prevents the varying depth of each billboard and results in inconsistency in rendering time. Therefore, we cull the rear-facing polygons and the polygons that are farther from the projecting plane than the limited depth value. Then, we cull the occluded polygons by using the occlusion query instruction in the graphics

library. Only the polygons that are not at all occluded are chosen.

The projected polygons may cover varying areas of the plane, which may result in a variety of billboard sizes. Therefore, we need to limit the size of enhanced billboards as well. In the implementation, we use only 2 image sizes, 512x512 and 256x256 pixels. We have a scale factor to scale the coordinates of the projected polygons to fit in the billboards. The scale factor scales the diagonals of the model's bounding box to a value of 512. Thus, it guarantees that all the polygons in the model can fit in the 512x512 pixel billboard. We also store the bounding box of each remaining polygon. The bounding box data is bound within an enhanced billboard and will be used in rendering time.

Build Enhanced billboards
Input: polygon model
Output: images representing enhanced billboards
Compute bounding box of the model
Set model-to-texture scale factor
Set of faces left F = input model
while $F > 0$
Choose direction with highest faces count
Cull back faces
Cull exceed depth value faces
Choose Texture resolution of output images
Cull occluded faces
Orthogonally project remaining faces
Save images as enhanced billboards

Figure 3: Pseudo-code of algorithm for construction of enhanced billboards

The remaining polygons can now be projected onto the plane of enhanced billboards using orthogonal projection. The free space on the billboard not covered by polygons is rendered as transparent pixels. The projected polygons will be discarded from the source model and the whole process is repeated until there is no polygon left. This results in a series of enhanced billboards representing some portions of the model from different viewing directions. By using all the enhanced billboards, we can reconstruct the complete model. Figure 3 shows the pseudo-code of the algorithm for the construction of enhanced billboards.

3.2 Rendering Enhanced Billboards

Because we treat an enhanced billboard as a height map, we can use a ray-height field intersection algorithm to render our enhanced billboards. The previous algorithm [Fab05] utilizes the programmability of the graphics hardware to do per-pixel ray-height field intersection. The height map image is rendered as a polygon quad. This method casts a ray from the eye to the texture coordinate in the height map polygon quad. The ray travels down the image and intersects at the first-hit position in the height map. The coordinate of the hit position is used to retrieve the color and normal information used in shading.

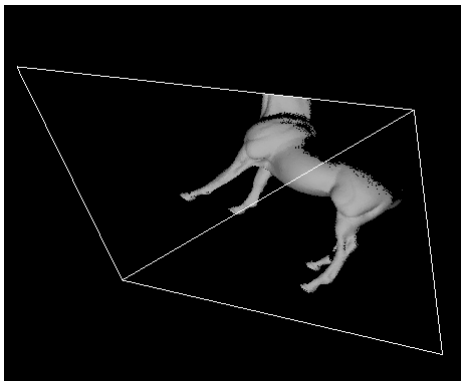


Figure 4: Rendering an enhanced billboard using only one plane. Some parts of the geometry are missing.

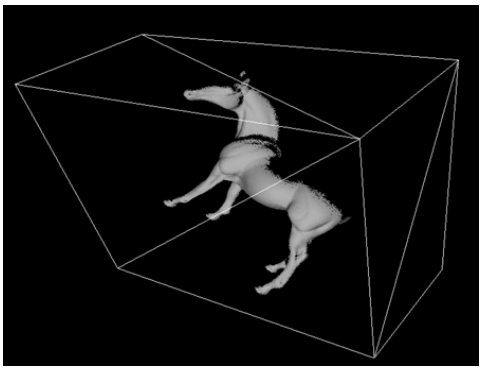


Figure 5: Rendering an enhanced billboard with all the bounding box planes. Thus, all the geometries are captured.

Unfortunately, previous algorithms could not produce an image with the silhouette feature; the boundary of the image will be flat. Furthermore, if we look at the image from the side view we will see a thin plane with nothing rendered. The earlier methods were built to render a patch of surface.

However, to render a complete model, many aspects of the algorithm had to be improved.

In order to correctly produce the silhouette, we use the bounding box information that comes with the enhanced billboard. For each enhanced billboard, we construct a bounding box of that enhanced billboard in screen space. Instead of rendering one plane with the previous ray casting algorithm, we render the enhanced billboard with all sides of its bounding box (see Fig.4 and Fig.5), which guarantees that all the geometry represented in the enhanced billboards will be rendered. In the view direction, three sides of the box are visible at most: the top and the two sides. The bottom of the box does not need to be rendered because the bottom plane shows the rear of the polygons.

The algorithm reports a hit if the depth of the traced ray is close to the depth value in the depth map. If the ray does not hit any position in the height map, the algorithm must report no hits (see Fig.6), and the pixel of the missed ray will not be rendered. Fig 7 shows the pseudo-code of the algorithm.

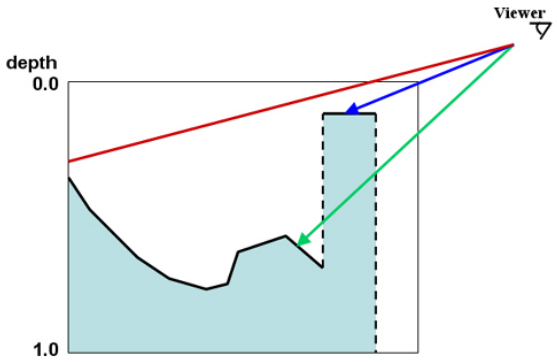


Figure 6: The red ray does not hit the depth map, thus reporting no hits. The green and blue rays hit the depth map at the arrow tips and report hits. Notice that the green ray passes through the dash line because only the black line represents the geometry while the dash line does not.

In the rendering algorithm, a ray is cast onto each pixel of the image. The algorithm performs a linear search to find the intersection of the ray and the depth map by stepping in the direction of the ray. We then compare the ray depth with the depth map. If the ray depth is in range (e = 0.01), the algorithm reports a hit. The step size is computed from $\sqrt{2}$ (diagonal of texture coordinate, which is the maximum length of the ray) divided by the total number of steps (user-defined).

```
Render Enhanced Billboards
Input: enhanced billboards images
Output: picture rendered on screen
e = hit range, maxStep = number of step
For each enhanced billboards
    Build bounding box of enhanced billboards
    For each texel on the plane of box
        start = texel position
        dir = direction from eye position to start
        d = 0; hitd = 0; hit = false;
        step = √2 / maxStep
        For 1 to maxStep
            ray = start + dir*d
            depth = GetDepthMapValue(ray.xy)
            if ( ray.z > depth) and (ray.z – e < depth)
                hit = true; hitd = d; exit loop
            d = d + step
        if (hit)
            hitpos = start + dir*hitd
            Render_Light(hitpos)
        else
            render as transparent pixel
```

Figure 7: Pseudo-code of algorithm for rendering ray-height field of enhanced billboards.

4. IMPLEMENTATION AND RESULT

We have implemented our rendering techniques described in the paper using HLSL. The programmability of the GPU is used for casting rays through enhanced billboards. Our testing system is a 2.8 GHz PC with 512 MB of memory, using a GeForce FX6800 with 128 MB of memory. We tested our method with a variety of polygon models, containing around 5,000 to 50,000 polygons.

The model was represented as enhanced billboards (see Fig.8 and Table 1.). The construction algorithm generated an output of around 8 to 10 enhanced billboards, each stored as 2 images. There might be a problem if the input model had two intersecting polygons, so the occlusion query was used to report that these two polygons were occluded and therefore not chosen. We dealt with this problem by forcing these intersecting polygons to bypass the occlusion test. There might be some polygons in the model which were so tiny that their projected areas were less than a pixel. We discarded these polygons. Some enhanced billboards that were generated from

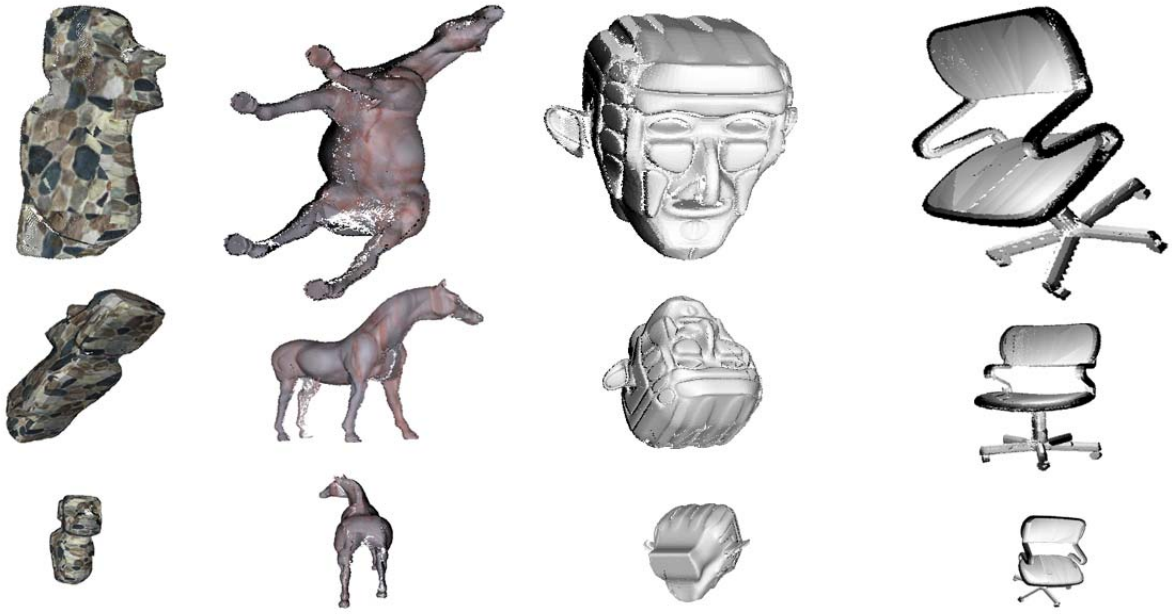


Figure 8: Variety of models rendered using enhanced billboards in different views and resolutions. Each of these images use around 4 – 5 enhanced billboards. Notice the correct silhouette. The render times are shown in Table 1.

Model	Polygons in Original Model	Build Time	Render Time High Resolution (800x600)	Render Time Medium Resolution (400x300)	Render Time Low Resolution (200x150)
Moai	5,000	26	7.11	15.31	35.36
Horse	40,000	195	10.02	18.34	40.12
Head	35,000	180	14.22	26.89	50.01
Chair	7,200	50	7.46	15.10	30.28

Table 1: The build times (seconds) and render times (fps) of different models. With different resolutions, enhanced billboards still represent the same amount of polygons. However, the frame rates are increasing because the models cover fewer pixels. Notice that the render time is not significantly dependent on the number of polygons in the original model.

the algorithm might contain little polygons seen as fragments of points in the image. To increase rendering performance, we also discarded these enhanced billboards because they had little contribution to the final rendered image. We decided not to project the polygons if they were almost perpendicular to the chosen direction even though they were facing forward. This was due to the fact that the enhanced billboard could not capture the information of almost perpendicular polygons. These polygons covered a small area of the enhanced billboard and thus were not sufficient for reconstructing the model.

Enhanced billboards are usually limited by the fill rates of the pixel shader, which uses a lot of pixel shader instructions. However, results have shown that enhanced billboards scale well with the pixels rendered. Thus, it was efficient in rendering objects at medium to high distance in any direction (see Fig 8.) with complete effects, such as parallax or silhouette. In contrast, the polygon representation methods did not scale according to the pixels they covered. Many distant polygons were rasterized to the same pixel, resulting in an alias in the rendered image.

5. CONCLUSION AND FUTURE WORK

We have presented a method for simplifying the model using a new representation and rendering technique. The enhanced billboards, each representing some portion of the model, store all the information of the original model. The information of the model is stored as a depth map, a normal map, a color map and a transparency map projected onto a plane by the greedy algorithm. Then, the enhanced billboards are rendered using the new ray-height field intersection algorithm. Our enhanced billboards can produce effects such as parallax and silhouette in any viewing directions. The quality of the enhanced billboard is comparable to the source model, and the frame rate does not depend on the number of polygons in the original model. Thus, this technique creates consistency in rendering time. Our result shows that the enhanced billboard is most suitable for rendering objects at medium to high distance where the pixels covered are less than those of the closer object.

We plan to improve the quality of enhanced billboards to solve the problem of missing fragments in the model, which occurs because some projected polygons have an area that is less than a pixel and are therefore discarded. Discarding consecutive polygons thus results in the missing fragments in the model.

The enhanced billboard technique can be easily extended to render shadow using the shadow map. The shadow map algorithm for enhanced billboards is not different from the polygonal model. First, the enhanced billboards are rendered from the light viewpoint and stored as a shadow map using the same rendering algorithm when viewing enhanced billboards. Then, the shadow map is used in rendering time the same way as in the polygonal model. Enhanced billboards can be used to speed up the ray intersection test for a ray-tracing algorithm. Usually, this algorithm builds an accelerated structure, such as a grid to query all the geometries occupied in the grid cell where the ray is located. However, enhanced billboards already represent the positions of all the geometries located inside the bounding box of the enhanced billboards. Thus, there is no need to build an auxiliary data structure.

Since enhanced billboards are stored as images, the image compression techniques can be used to lower the size of image files. In recent years, wavelet compression has gained popularity [Ren03] due to its compression factor combined with minimal loss of overall image feature. Enhanced billboards can benefit from using wavelet compression to lower the amount of storage needed.

6. REFERENCES

[Bax02] Baxter, B., Sud, A., Govindaraju, N., and Manocha, D. Gigawalk:Interactive walkthrough of complex 3d environments, *Proc. of Eurographics Workshop on Rendering*, 2002

[Bli78] Blinn, J. Simulation of Wrinkled Surfaces, *SIGGRAPH 78*, pp. 286-292, 1978

[Fab05] F´abio Policarpo, M.M.O., Jo˜ao L. D. Comba Real-Time Relief Mapping on Arbitrary Polygonal Surfaces, *ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games 2005*, 2005

[Gar97] Garland, M., and Heckbert, P. Surface simplification using quadric error bounds, *Pro. of ACM SIGGRAPH*, pp. 209-216, 1997

[Hir04] Hirche, J., Ehlert, A., Guthe, S.,and Doggett, M. Hardware accelerated per-pixel displacement mapping, *Graphics Interface*, pp. 153 – 158, 2004

[Hop04] Hoppe, H. Geometry clipmaps: Terrain rendering using nested regular grids, *ACM SIGGRAPH 2004*, pp. 769-776, 2004

[Hop96] Hoppe, H. Progressive meshes, *ACM SIGGRAPH 1996*, pp. 99-108, 1996

[Hop97] Hoppe, H. View dependent refinement of progressive meshes, *ACM SIGGRAPH Conference Proceedings*, pp. 189-198, 1997

[Jak00] Jakulin, A. Interactive vegetation rendering with slicing and blending, *Proc. Eurographics 2000 (short papers) (Aug. 2000)*, 2000

[Jes02] Jeschke, and Wimmer, M. Textured depth mesh for real-time rendering of arbitrary scenes, *Proc. Eurographics Workshop on Rendering*, 2002

[Mey98] Meyer A, N.F. Interactive volumetric textures, *Eurographics Rendering Workshop*, 1998

[Mey01] Meyer A., N.F., POULIN P. Interactive rendering of trees with shading and shadows, *Eurographics Workshop on Rendering (Jul 2001)*, pp. 183-196, 2001

[Oli00] Oliveira, M.M., Bishop, G., and Mcallister, D. Relief texture mapping, *Siggraph 2000, Computer Graphics Proceedings*, pp. 359-368, 2000

[Per97] Peercy, M., Airey, J., and Cabral, B. Efficient bump mapping hardware, *SIGGRAPH '97*, pp. 303-306, 1997

[Pha96] Pharr, M., and Hanrahan, P. Geometry caching for ray-tracing displacement maps, *Eurographics Rendering Workshop 1996*, pp. 31-40, 1996

[Phi04] Philippe Decaudin, F.N. Rendering Forest Scenes in Real-Time, *Eurographics Rendering Workshop 2004*, 2004

[Ren03] Ren Ng, R.R., Pat Hanrahan. All-Frequency Shadows Using Non-linear Wavelet Lighting Approximation, *SIGGRAPH 03*, 2003

[Sch99] Schaufler, G., and Priglinger, M. Efficient displacement mapping by image warping, *Eurographics Rendering Workshop 1998*, pp. 175-186, 1999

[Sha98] Shade, J.W., Gortler, S. J., HE, L.-W., andSzeliski, R. Layered depth images, *Siggraph 1998, Computer Graphics Proceedings*, pp. 231-242, 1998

[Smi00] Smits, B.E., Shieley, P., and Stark, M. M. Direct ray tracing of displacement mapped triangles, *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pp. 307-318, 2000

[Wan03] Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., and Shum, H.-Y. View-dependent displacement mapping, *ACM Trans. Graph. 22*, pp. 334-339, 2003

[Xav03] Xavier Decoret, F.D., Franc¸ois X. Sillion, Julie Dorsey. Billboard Clouds for Extreme Model Simplification, *SIGGRAPH 03*, 2003

[Xi04] Xi Wang, X.T., Stephen Lin, Shimin Hu, Baining Guo, Heung-Yeung Shum. Generalized Displacement Maps, *Eurographics Rendering Workshop 2004*, 2004

A Novel Technique for Opus Vermiculatum Mosaic Rendering

S. Battiato G. Di Blasi G.M. Farinella G. Gallo

IPLab – Image Processing Laboratory
http://www.dmi.unict.it/~iplab/
Dipartimento di Matematica e Informatica
University of Catania, Via Andrea Doria 6 – 95125, Catania (Italy

{battiato, gdiblasi, gfarinella, gallo}@dmi.unict.it

ABSTRACT

In this paper we present a method to generate a digital mosaic starting from a raster input image. Mosaics generation of artistic quality is challenging. The basic elements, the tiles, typically small polygons, must be packed tightly, emphasizing orientations chosen by the artist. An ad-hoc boundaries detection have to be performed according to the directional guidelines. Different mosaic styles can be automatically rendered, depending on artistic techniques considered ("opus musivum", "opus vermiculatum", etc.).
The proposed method is able to reproduce the colors of the original image emphasizing relevant boundaries by placing tiles along their direction. The boundaries detection is based on the statistical region merging algorithm. In particular the technique is able to reproduce the "opus vermiculatum" mosaic style.
Several examples reported in the paper show how the right mixture of mathematical tools together with century proved ideas from mosaicists may lead to impressive results.

Keywords
mosaic, non-photorealistic rendering, distance transform, image processing and enhancement, statistical region merging.

1. INTRODUCTION

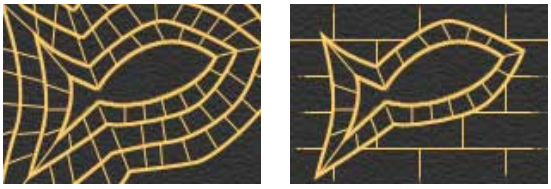
Mosaics are artworks constituted by cementing together small colored tiles. Probably, they are the first example of image synthesis techniques based on discrete primitives. A smart and judicious use of orientation, shape and size may allow to convey much more information than the uniform or random distribution of *N* graphic primitives (like pixels, dots, etc.). For example, ancient mosaicists avoided lining up their tiles in rectangular grids, because such grids emphasize only horizontal and vertical lines. Such artifacts may distract the observer from seeing the overall picture described. To overcame such potential drawback, old masters placed tiles emphasizing the strong edges of the main subject to be represented, as shown in Figure 1.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

Figure 1. Example of mosaic.

In [Dib05a] a technique able to render traditional looking mosaics is presented. As the authors point out, the main problem to solve in the mosaic rendering is the directional guidelines detection. Directional guidelines are related with the salient edges of the image and the unsupervised detection is, "per se", an interesting and challenging problem. Moreover, directional guidelines and edges are two related but different features. The use of classical edge detector algorithms (for example [Mee01]) cannot hence be transferred without great care to the problem at hand.
The authors present also a simple technique to detect the directional guidelines, but they grant that although they have adopted a practical solution that is satisfactory for the application, the problem deserves a

theoretical and algorithmic deeper investigation.
In this paper we present another directional guidelines detection technique based on the Gestalt movement idea of the perceptual grouping [Kof22]. In particular, the statistical region merging algorithm [Noc04] has been used. This method allows a more precise detection of the most important feature curves of an image, taking into account simple statistical consideration in a pixel neighborhood.
We extend the technique presented in [Dib05a] in order to produce another mosaic style called "opus vermiculatum" (see Figure 2).



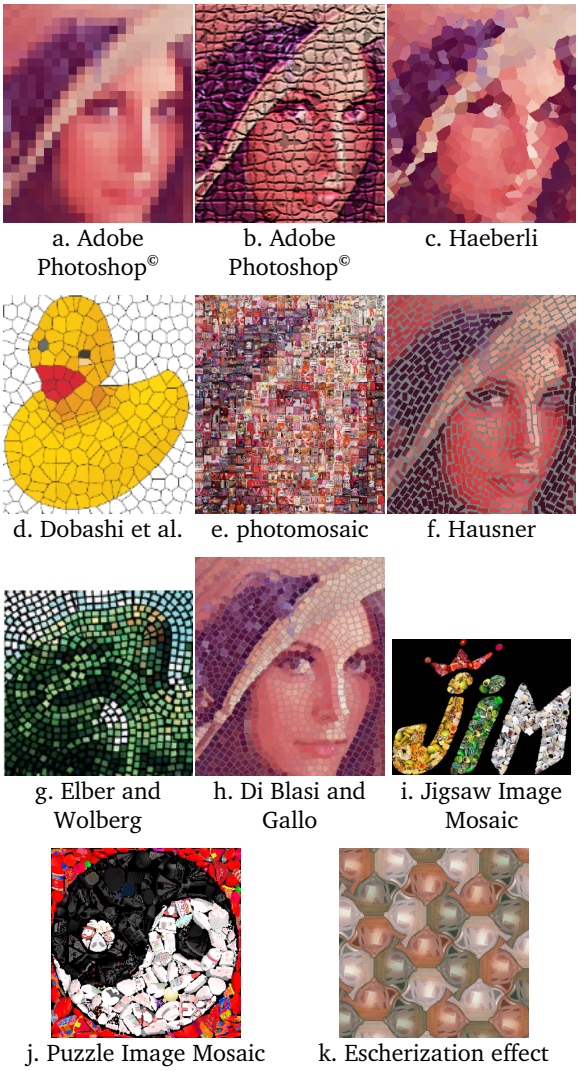
a. Opus musivum b. Opus vermiculatum

Figure 2. Examples of ancient mosaics stiles.

As the epithet of "opus musivum" says this means of "quality worth of the muses", of great visual refinery and effect. "Opus vermiculatum" takes its name from the Latin for "worm". It refers to lines of tiles that snake around a feature in the mosaic. Often two or three rows of "opus vermiculatum" appear like a halo around something in a mosaic picture, helping it stand out from the background (see Figure 1).
The rendering of "opus vermiculatum" mosaics requires a clear separation between foreground and background because the two regions of the image have to be managed in different ways. The foreground region is covered as an "opus musivum" (see [Dib05a] for details), while the background region have to be covered by a regular grid of tiles (eventually perturbed by a random noise in size, position and rotation).
The rest of the paper is organized as follows: in Section 2 we summarize previous related works, Section 3 explains the details of the proposed algorithm. In Section 4 some experimental results are reported. Finally in Section 5 we suggest directions for future works and research.

2. REVIEW OF PREVIOUS WORKS

Computer Graphics attempts to simulate mosaics inscribe themselves into the broader area of non-photorealistic rendering (NPR).
Although mosaics are a traditional art form attempts to simulate them in the digital realm are recent. Commercial image processing software provide "mosaic filters" to obtain tessellated images (the examples in Figure 3a and Figure 3b have been produced with Adobe Photoshop© [Ado06]).
More sophisticated approaches try to adopt smart strategies using computational geometry together with image processing tricks.
Haeberli [Hae90] used Voronoi diagrams, placing the sites at random and filling each region with a color sampled from the image. This approach tessellates the image, but tile shapes are too variable and do not attempt to follow edge features (see Figure 3c). This technique is also available in many user-end applications under the name of "crystallization" and it simulates the typical effect of some glass windows in the churches. In [Dob02] Dobashi et al. reprised the Haeberli's idea obtaining good results (see Figure 3d). They use an optimization technique based on the edge features of the image which avoids to place a tile across a meaningful border of the picture.



a. Adobe Photoshop© b. Adobe Photoshop© c. Haeberli
d. Dobashi et al. e. photomosaic f. Hausner
g. Elber and Wolberg h. Di Blasi and Gallo i. Jigsaw Image Mosaic
j. Puzzle Image Mosaic k. Escherization effect

Figure 3. Mosaic effects.

"Photomosaic" [Sil97] transforms an input image into a rectangular grid of thumbnail images (see Figure 3e). In this approach the algorithm searches in a large database of images to find the best approximation of a block of pixels in the main image. The resulting effect is very impressive, but even in this case no edge features are respected. The idea was successively extended by Klein et al. [Kle02] to videos obtaining a video mosaic. Recently Di Blasi and Petralia [Dib05b] presented an approach to speed up the search process

based on the Antipole strategy [Can05]; the same technique has been previously used for fast texture synthesis [Bat03] and fast colorization of gray images [Dib03].
Hausner [Hau01] obtains very good results using centroidal Voronoi diagrams, edge features, L_1 (Manhattan) distance and graphic hardware acceleration to optimize the results (Figure 3f). A very advanced approach to the rendering of traditional mosaics is presented in [Elb03]. This technique is based on offset curves that get trimmed-off the self intersecting segments with the guidance of Voronoi diagrams. The algorithm requires a mathematical description of the edges (e.g. B-splines) allowing an accurate tile placement (Figure 3g). Other advantages of this approach is the use of variable size tiles. Although the results are very good the technique seems limited to the case of a single, user-selected and close edge curve. As previously mentioned another approach for the generation of ancient mosaics is presented in [Dib05a]; this approach is based on directional guidelines, distance transform, mathematical tools and century proved ideas from mosaicists leading out impressive results (Figure 3h).
Kim and Pellacini [Kim02] introduce a mosaicing technique where image tiles of arbitrary shapes are used to compose the final picture. The idea is quite similar to the photomosaic, but the final effect is very different and interesting (Figure 3i). Another approach for the creation of the same kind of mosaics is presented in [Dib05c]; the described approach leads to impressive results in an acceptable computation time (Figure 3j).
For sake of completeness we also cite "Escherization" [Kap00], a technique that produces tilings of the plane using slightly distorted version of an image (Figure 3k). It relies on symmetry groups and regular tilings. It is very different from the other kind of methods we reviewed above and it is aimed to the production of a sophisticated kind of aesthetic effects different than mosaics.

3. THE PROPOSED ALGORITHM

In this Section we present our mosaic algorithm describing the segmentation algorithm used to distinguish interactively foreground and background areas together with the details needed to obtain the "opus vermiculatum" mosaics.

3.1 Statistical Perceptual Grouping Segmentation for Guidelines Detection

Segmentation is a process useful to subdivide an image into its meaningful regions, corresponding to objects represented in a scene. An image segmentation allows an easy guidelines detection: boundaries of each detected region are considered as guidelines. The precision of the segmentation is a decisive factor for a good guidelines detection.
Gestalt movement [Kof22] claims that the perceptual grouping has a fundamental role in human perception. Following this idea, the visual ability is reformulated as a statistical inference problem.
Statistical Region Merging [Noc04] (SRM) is a recent segmentation technique able to capture the main structural components of a digital image using a simple but effective statistical analysis.
SRM is based on two components:

1. *Statistical Merging Predicate*, used to establish if two regions have to be merged;
2. *Merging Order*, used to establish the order to test the Statistical Merging Predicate;

The pseudo-code of the algorithm is reported below:

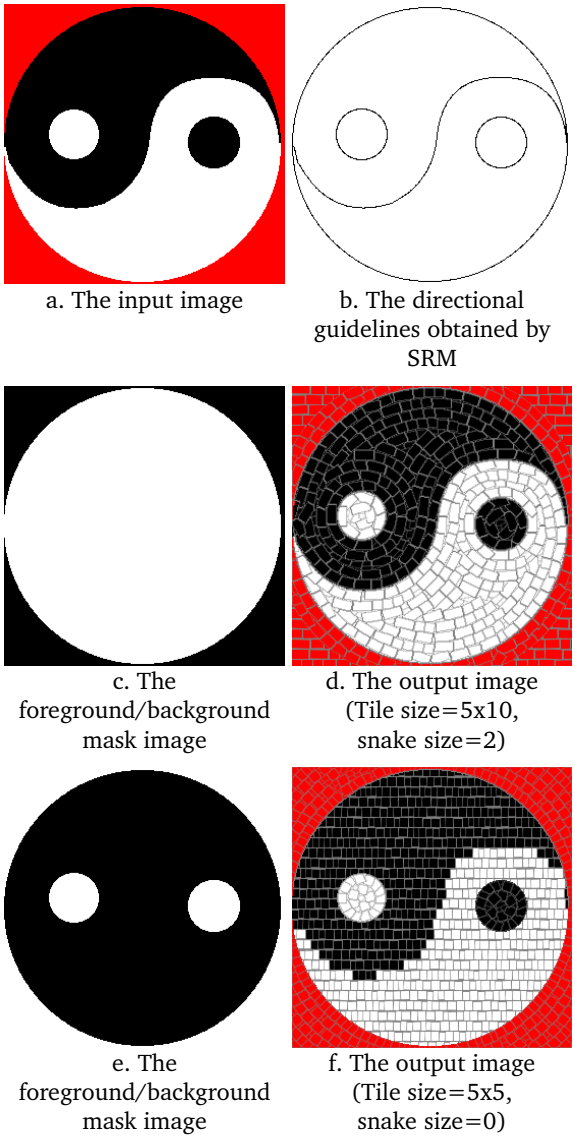
```
Input: an image I

Let S_I be the set of the 4-connexity
couples of adjacent pixel in I.

S'_I=Order_increasing(S_I, f);

For i=1 to |S'_I| do

  If R(p_i)≠R(p'_i) and P(R(p_i),R(p'_i)) = true
     then Union(R(p_i), R(p'_i))

end For
```

$f(p, p')$ is a real-valued function, with p and p' adjacent (4-connexity) pixels in I. f is used to establish the Merging Order (i.e. $Order\_increasing(S_I, f)$). The function f used in our case approximates the following invariant: "when any test between two true regions occurs, that means that all tests inside each of the two true regions have previously occurred".
An important tuning parameter of SRM is the merging factor; it allows to control the coarseness of the segmentation. In our case we use a high merging factor value to force an explicit over-segmentation needed for our purposes (as described in Section 3.2).
It is straightforward to verify that the time asymptotic complexity is linear in the number of pixels of the image.
SRM has been recently successfully used for microarray analysis ([Bat06b]). Experimental results show that SRM can be effectively used to solve the guidelines detection problem.

3.2 Rendering of Opus Vermiculatum Mosaics

In order to produce an "opus vermiculatum" mosaic, an "a priori" segmentation of the input images is needed: as previously stated such artistic mosaic is characterized by the different styles used to place tiles in the foreground and in the background. In the foreground, tiles are placed as in the "opus musivum" mosaic, while in the background tiles have to be placed in a regular grid, eventually perturbed by a random noise in size, position and rotation.
Moreover, foreground region have to be partially expanded in order to simulate the lines of tiles that snake around the foreground features (often two or three rows).
It could be possible to use some algorithms able to automatically find foreground and background regions

in an image (see for example [Kim05] and [Pic04]): these methods are often imprecise and noise dependent giving out, for our purposes, poor and unuseful information. Moreover, we need also a description of the details inside the foreground areas. For these reasons we preferred using a semi-automatic technique based on the SRM method.
The algorithm first automatically segments the image using a high merging factor, this leads to a very rough (over-merged) segmentation, but sufficiently precise for our aims. Second, an user-friendly GUI allows the user to easily select the foreground and the background regions.



a. The input image

b. The directional guidelines obtained by SRM

c. The foreground/background mask image

d. The output image (Tile size=5x10, snake size=2)

e. The foreground/background mask image

f. The output image (Tile size=5x5, snake size=0)

Figure 4. Some examples of opus vermiculatum mosaic.

This leads to a detailed foreground/background mask image which can be used in the successive steps. Note that this approach allows the user to produce several foreground/background mask images from the same input.

Then, the algorithm creates an "opus musivum" for the foreground regions and a regular grid of tiles for the background. Finally, it merges the two images in order to produce the "opus vermiculatum" of the input image. It is possible to perform an "a-priori" morphological operation of erosion on the mask image in order to produce the snake effect around the foreground features.
Figure 4 shows an example of the proposed technique; in particular Figures 4d and 4f show how the user can easily modify the final result by a different choice of the foreground/background mask image (Figures 4c and 4e, respectively) and of the snake size.

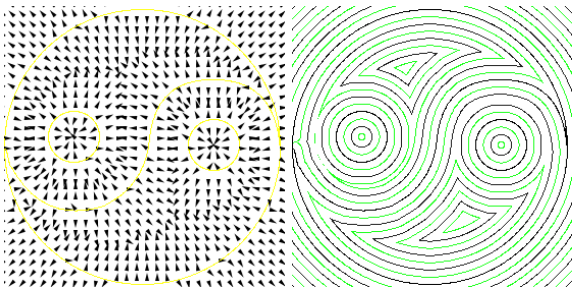
3.3 Foreground Rendering of Opus Musivum Mosaics

In this Subsection we briefly present the technique able to create "opus musivum" mosaics. Using the directional guidelines the algorithm first evaluates for each pixel of the image the distance transform [Har92], i.e. its minimum distance from any guideline pixel, obtaining a matrix (*dtM*). The use of the distance transform in the field of NPR was previously proposed by Gooch et al [Goo02], for a different purpose.
Starting from the distance transform matrix it obtains another two matrices: the gradient matrix (*gM*) and the level line matrix (*llM*):

$$gM(x,y)=\arctan\frac{dtM(x,y+1)-dtM(x,y-1)}{dtM(x+1,y)-dtM(x-1,y)} \quad (1)$$

$$llM(x,y)=\begin{cases}1 & if\ module(dtM(x,y),2\cdot tSize)=0\\ 2 & if\ module(dtM(x,y),2\cdot tSize)=tSize\\ 0 & elsewhere\end{cases} \quad (2)$$

where *tSize* is an user-selected integer value that denotes the size for the tiles. Their meaning is clearly shown in Figure 5 (in Figure 5b, black pixels have value 1, green pixels have value 2).



a. The gradient matrix

b. The level line matrix

Figure 5. Visual representation of the matrices used by the algorithm.

The algorithm is ready to place the tiles using the pixels in *llM* with value 2. Observe that such pixels form chain-like sequences. More precisely the algorithm proceeds as follows:

- while there are chains of pixels with value 2 not

yet processed:

a. select a chain;
b. starting from an arbitrary pixel on it "follow" the chain;
c. place new tiles at regular distances along the path (the orientation of the tiles is assigned using the gradient information from matrix *gM)*.

The distance along the chain that separates successive tiles is equal to *sSize* when tiles of dimension *tSize*x*sSize* have been adopted.

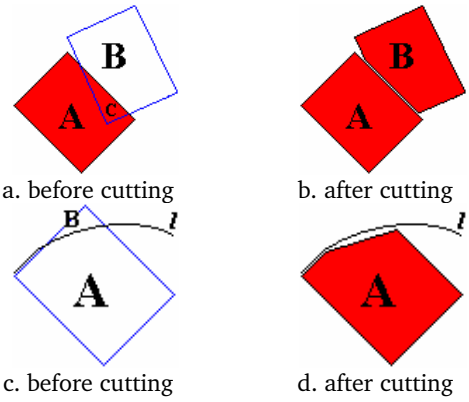
If tiles of fixed size and shape are positioned only according to the method described insofar two main difficulties arise:

1. tiles may overlap;
2. a single tile may cover an area across the "black pixels lines" (i.e. the pixels with value 1 in *llM*).

Both of these effects are unpleasant. In particular the problem in 2., completely destroys the guideline patterns and would result into blurred images.

To address these difficulties the algorithm adopts the following heuristic strategy:

1. the overlapping of tiles is easily detected maintaining a boolean mask of covered pixels. If a tile to be placed contains pixels already covered by previously placed tiles we change the original rectangular shape of the tile "cutting away" the overlapping pixels (see Figures 6a and 6b);
2. if a new tile crosses any "black pixel line" it is trimmed against this line (see Figures 6c and 6d).



a. before cutting b. after cutting

c. before cutting d. after cutting

Figure 6. Tiles cutting methodology.

Once the tile positioning and cutting phase has been completely carried out a couple of post-processing steps have to be performed in order to achieve a pleasant aesthetic effect.

First, "grout spaces" between tiles are important. To achieve the effect of cement showing through tiles a downscaling of each tile is done. This frees some pixels that will be assigned a unique color for concrete under the mosaic.

Second, the algorithm chooses for each tile to have a uniform color equal to the color of the pixel corresponding to its center in the source image. Other choices may lead to different artistic effects (for example a random perturbation of the selected color).

Major details can be found in [Dib05a].

4. EXPERIMENTAL RESULTS

To illustrate the effectiveness of the proposed technique we report both pictorial examples and some quantitative results. The algorithm has been implemented in Java2 Standard Edition 1.4.2 and all experiments have been carried out on a PC Athlon XP-M 1800+, 192MB RAM, with Windows XP Home Edition. To allow the real testing of the performances of the proposed technique an applet is available [Bat06a], at the same URL is also possible to download a JGimp plug-in and a Java application. Our implementation at this time does not rely on any particular graphic hardware acceleration.

Different parameters choices can lead to different final results. In our experiments, we used a merging factor for SRM equals to 3, a kernel size for the erosion morphological operation proportional to the number of lines of tiles snaking around the foreground and a Laplacian 3x3 kernel filter for the directional guidelines detection.

A mosaic made up of *N* tiles conveys more information than an image made up of *N* rectangular pixels in a regular lattice. Examples are shown in Figure 7 and 8. The proposed technique takes as input a digital image without any limitations on colors and resolution size. We found very appealing the results obtained using modern art sources like Cézanne's painting and Antonello da Messina artworks (Figures, 7c, 7d and 8, respectively). Figures 7a and 7b prove the effectiveness of the proposed technique when the source image is a good quality photograph.

Timing results (Table 1) show how the algorithm is fast enough to be used as a plug-in in a typical user-end software. It is straightforward to verify that the time asymptotic complexity is linear in the number of pixels of the image.

Size	Mosaic Mean Time (sec.)
600x600	10.485
800x600	12.689
593x886	15.182
1024x768	22.142

Table 1. Timing results.

5. CONCLUSIONS AND FUTURE WORKS

In this paper we presented a new method to create artificial mosaics achieving an impressive visual impact. Experimental results show the soundness of our algorithm.

There are several ways to improve the aesthetic of our results and several ideas started from this work:

1. a different strategy for choosing, in case of tile overlapping, which tile has to be cut. Heuristic rules or, perhaps, randomized choices could produce different outcomes;
2. automatic optimized choices of tile scale relative

to each input image is an open problem worth of further investigations;

3. generalization of our “mosaicists' heuristic” to other kind of primitive based on NPR processing seems possible and quite promising;
4. some generalizations as proposed in [Elb03], such as variable size tiles and photomosaic, are also considered for future work and research; it is also interesting to explore the possibilities offered using different basic shapes than rectangular tiles;
5. a different method to better find the directional guidelines is an important research investigation issue (see for example extensions proposed in [Noc05]);
6. exploitation of hardware graphics primitives to accelerate the mosaic synthesis;
7. extension of our method for mosaic rendering of 3D surface is probably the most exciting direction of research.

6. REFERENCES

[Ado06] Adobe Photoshop. http://www.adobe.com, 2006

[Bat03] S. Battiato, A. Pulvirenti, D. Reforgiato. Antipole Clustering for Fast Texture Synthesis. In proceedings of ACM/WSCG2003, 2003

[Bat06a] Battiato S., Di Blasi G., Farinella G.M., Gallo G. The Artificial Mosaic Creator applet www.dmi.unict.it/~gdiblasi/mosaic/mosaic.html, JGimp plug-in and Java application www.dmi.unict.it/~gdiblasi/mosaic/mosaic.jar, 2006

[Bat06b] Battiato S., Di Blasi G., Farinella G.M., Gallo G., Guarnera G.C. Ad-Hoc Segmentation Pipeline for Microarray Image Analysis. In proceedings of IS&T/SPIE Electronic Imaging 2006, 2006

[Can05] Cantone D., Ferro A., Pulvirenti A., Reforgiato Recupero D., Shasha D. Antipole Tree indexing to support range search and K-nearest neighbor search in metric spaces. IEEE Transactions on Knowledge and Data Engineering 17 (4), pp. 535-550, 2005

[Dib03] Di Blasi G., Reforgiato Recupero D. Fast Colorization of Gray Images. In proceedings of Eurographics Italian Chapter 2003, 2003

[Dib05a] Di Blasi G., Gallo G. Artificial Mosaic. The Visual Computer 21 (6), pp. 373-383, 2005

[Dib05b] Di Blasi G., Petralia M. Fast Photomosaic. In poster proceedings of ACM/WSCG2005, 2005

[Dib05c] Di Blasi G., Gallo G., Petralia M. Puzzle Image Mosaic. In proceedings of IASTED/VIIP2005, 2005

[Dob02] Dobashi J., Haga T., Johan H., Nishita T. A Method for Creating Mosaic Images Using Voronoi Diagrams. In proceedings of Eurographics2002, pp. 341-348, 2002

[Elb03] Elber E., Wolberg G. Rendering Traditional Mosaics. The Visual Computer, 19 (1), pp. 67-78, 2003

[Goo02] Gooch B., Coombe G., Shirley P. Artistic Vision: Painterly Rendering using Computer Vision Techniques. In proceedings of NPAR, pp. 83-90, 2002

[Hae90] Haeberli P. Paint by Numbers. In proceedings of SIGGRAPH1990, pp. 207-214, 1990

[Har92] Haralick R., Shapiro L. Computer and Robot Vision - Vol. 1. Addison-Wesley Publishing Company, 1992

[Hau01] Hausner A. Simulating Decorative Mosaics. In proceedings of SIGGRAPH2001, pp. 573-580, 2001

[Kap00] Kaplan C., Salesin D.H. Escherization. In proceedings of SIGGRAPH2000, pp. 499-510, 2000

[Kim02] Kim J., Pellacini F. Jigsaw Image Mosaics. In proceedings of SIGGRAPH2002, pp. 657-664, 2002

[Kim05] Kim K., Chalidabhongse T.H., Harwood D., Davis L. Real-time foreground–background segmentation using codebook model. Real-Time Imaging 11(3), pp 172-185, 2005

[Kle02] Klein A.W., Grant T., Finkelstein A., Cohen M.F. Video Mosaics. In proceedings of NPAR2002, pp. 21-28, 2002

[Kof22] Koffka, K. Perception: and introduction to the Gestalt-theorie. Psychological Bulletin 19, pp. 531-585, 1922

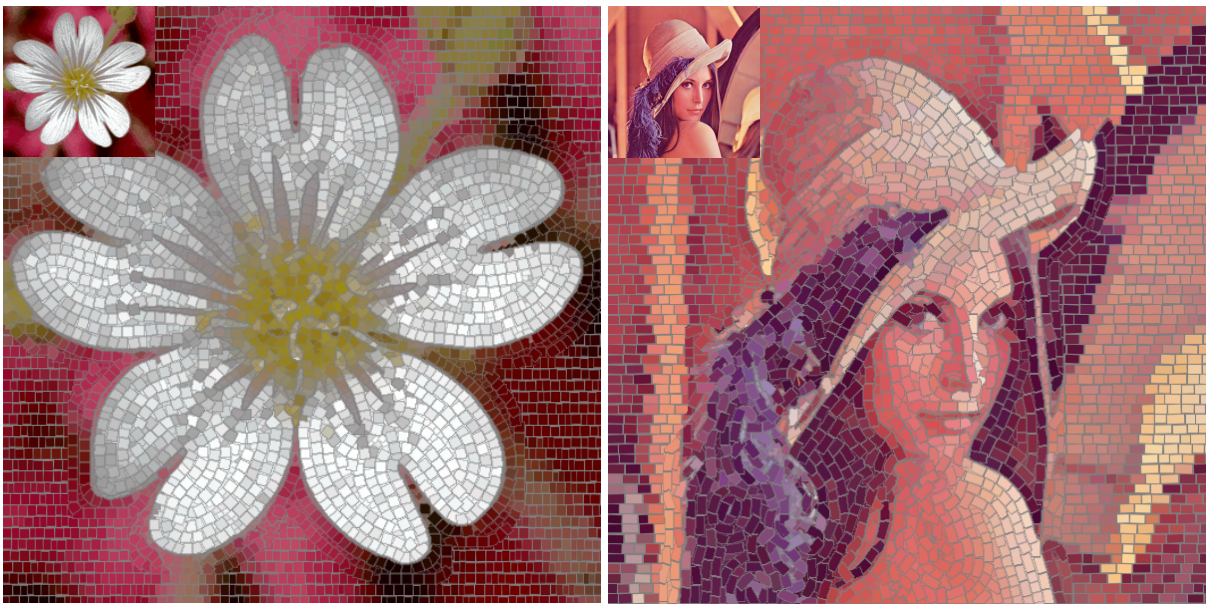
[Mee01] Meer P., Georgescu B. Edge Detection with Embedded Confidence. IEEE Transaction on Pattern Analysis and Machine Intelligence 23 (12), pp 1351-1365, 2001

[Noc04] Nock R., Nielsen F. Statistical Region Merging. IEEE Transaction On Pattern Analysis and Machine Intelligence 26 (11), pp. 1452-1458, 2004

[Noc05] Nock R., Nielsen F. Semi-supervised statistical region refinement for color image segmentation. Pattern Recognition 38 (6), pp. 835-846, 2005

[Pic04] Piccardi M., Jan T. Mean-shift background image modelling. In proceedings of ICIP2004, pp. 3399- 3402, 2004

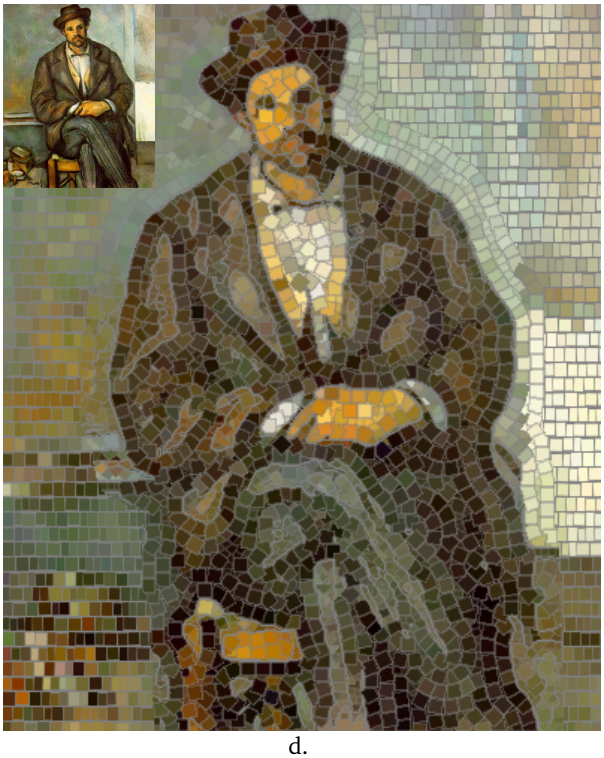
[Sil97] Silvers R., Hawley M. Photomosaics. Henry Holt, New York, 1997



a.

b.

c.



d.

Figure 7. Some examples of our algorithm applied on two typical test images (*a.* and *b.*) and two real well known Cézanne paintings (*c.* and *d.*).

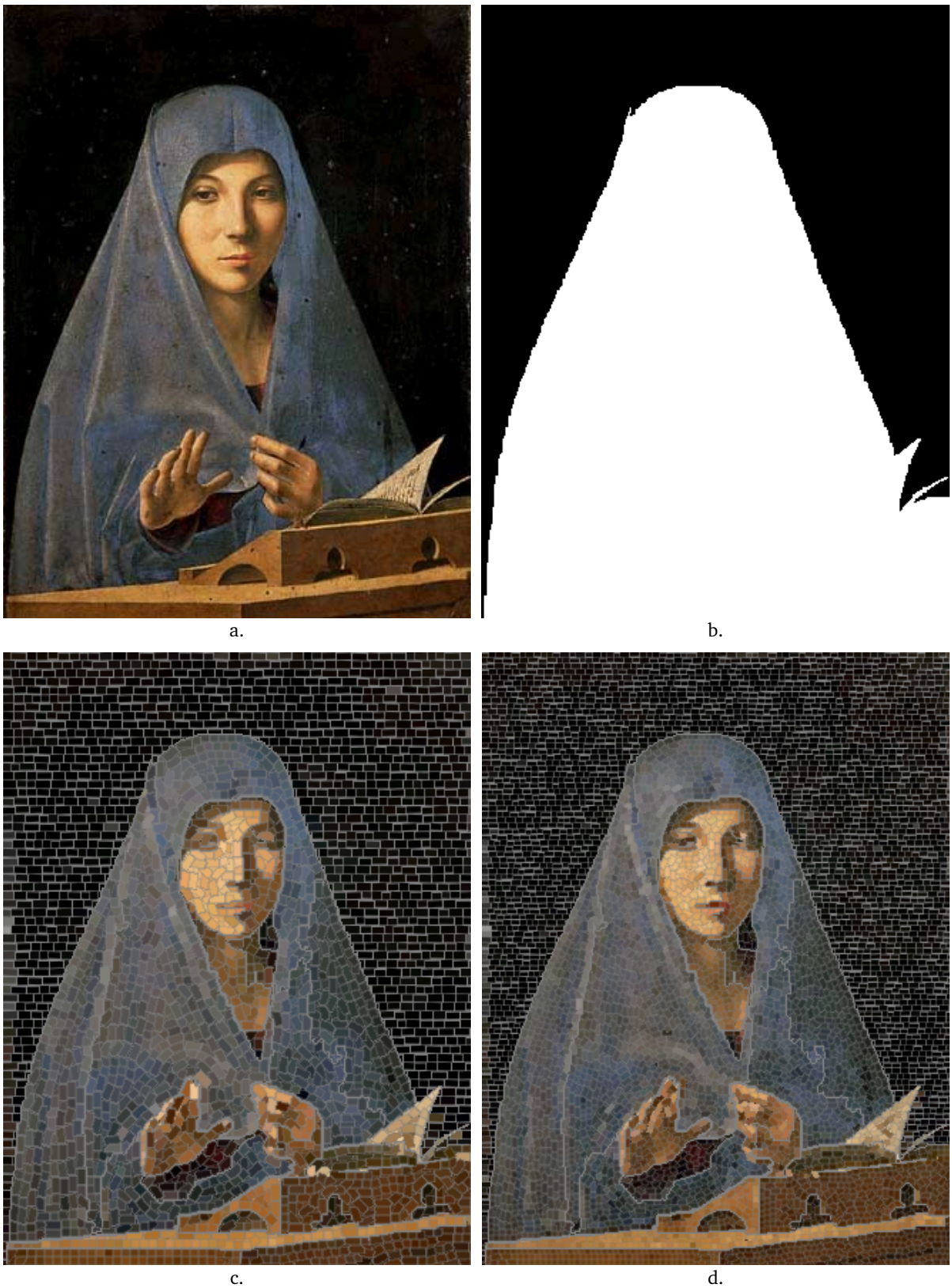


Figure 8. The Antonello da Messina painting *a.*, the background/foreground mask *b.* together with two different mosaics obtained using respectively the tile size 5x10 on *c.* and 3x6 on *d.*.

2D Multilayer Painterly Rendering with Automatic Focus Extraction

Levente Kovács
University of Veszprém
Dept. of Image Processing and Neurocomp.
Egyetem u. 10.
H-8200, Veszprém, Hungary
kla@vision.vein.hu

Tamás Szirányi
Hungarian Academy of Sciences
Comp. and Automation Research Institute
Kende u. 15.
H-1111, Budapest, Hungary
sziranyi@sztaki.hu

ABSTRACT

We present an automatic two dimensional model based non-photorealistic painterly rendering method which uses automatic relative focus map extraction from the model image to produce a relevance-based multilayer painting. Using relative focus segmentation, the painterly rendered image will be built of differently detailed layers.

Keywords

stroke based rendering, artificial painting, focus based painting

1. INTRODUCTION

Non-photorealistic model based 2D painterly rendering has produced quite a large family of techniques over the years. Many of these methods, which we will call classical, generate images in an automatic, semi-automatic or manual process with the goal of producing visually pleasant painting-like renderings of real life model images. Most of these techniques use some form of artificial stroke model to generate an artificial painting on a virtual canvas.

There are many ways of generating painting-like images from ordinary pictures. The rough outline of these algorithms is as follows. We take a model image, and a blank image called canvas, on which we will create the painterly rendered image. This will consist of a series of brush strokes that will be placed on the canvas. The strokes can have different size, shape, orientation and color. These properties are the so-called stroke parameters which are the data needed for later reconstruction. Strokes can be very versatile. There are methods which use simple shapes as strokes, others that use long curves with constant or varying width, or use different templates or texture patterns as stroke models. The methods differ at most in the way they place the strokes on the canvas in order to obtain a plausible artistic representation of the model image.

FULL Papers conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

The method we present in this paper introduces new tools in the painterly image generation process and based on these tools we position the presented method in the family of so called alternative painterly rendering techniques. Alternative in the sense that a wide range of outsider – from the point of view of computer graphics – tools is incorporated in order to improve the painterly rendering process. In our case such tools from the domain of image analysis are an automatic relative focus map extraction method to extract more relevant areas of the model images, and an image segmentation method for background replacement and color extraction. Using these tools our stroke based rendering method can produce a two layer rendering with less detail for the background areas with finer painting over the relevant areas.

Classical 2D painterly rendering techniques include e.g. the work of Haeberli [Hae90a] who introduced a painting method by following the cursor, point sample the underlying color and paint a stroke on the current position with that color, the process being controlled by a stochastic process. Hertzmann's algorithm [Her98a] painted an image with a series of multilayer B-spline modeled strokes on a grid over the canvas. Szirányi et al. introduced the so-called Paintbrush Image Transformation [Szi00a] which was a simple random painting method using rectangular

stroke templates, up to ten different stroke scales in a coarse to fine multilayer way also for segmentation and classification purposes.

Among alternative techniques, in the work of Gooch et al. [Goo02a] a method was presented that produced a painting-like image composed of strokes by first segmenting the image into features, finding the approximate medial axes of these features, and using the medial axes to guide brush stroke creation. Park et al. presented a brush generation technique where spline brush colors are obtained by generating a color palette from selected reference images [Par04a]. Kovács et al. presented cartoon-style rendering techniques based on stochastic painting with stroke templates with automatic multiscale painting on painted images and image sequences [Kov04a] was shown and they also suggested effective storage and compression possibility of real life paintings processed with stochastic paintbrush transformation [Kov04b]. Santella and DeCarlo presented a method [Dec02a] which combines aspects from the approaches of Haeberli [Hae90a], Litwinowicz [Lit97a] and Hertzmann [Her98a]. They extended their work with a system collecting eye-tracking data from a user [San02a].

The latter technique of Santella and DeCarlo was also an inspiration to the present work. The relative focus map extraction technique [Kov05a] which we use in this work makes possible to automatically extract relevant regions of the model images and run the painterly rendering with giving more detail on those areas and less for the others.

2. PAINTERLY RENDERING

The method we describe in this Section is a multilayer stroke based rendering technique. It is based on a modified version of the painterly transformation method in [Kov04b]. It uses two layers of painting and three stroke scales for generating the painterly output. The main steps of the method are the following:

1. Take input image A, blank canvas C.
2. Generate edge map E of model A for edge orientation data, by scale-space edge extraction [Lin98a], for extracting weighted edge map used during the painting process like in [Kov04a].
3. Generate relative focus map F of model A for relevant region extraction [Kov05a].
4. Generate segmentation map for extracting similarly colored areas (described below).
5. Generate the background layer
   a. with the two large stroke scales (60x15 and 32x8 masks, in this case with rectangular stroke templates), with stroke orientations determined from E.
   b. by using the colorized segmentation map as low-detail background.
6. Refine the areas from F with the small scale stroke (10x3 mask).

The input image A will be the model for the rendering process, while the canvas C is a blank image as a starting point. The edge map is generated with scale space edge extraction [Lin98a] for obtaining the strongest edges which will give the orientation of the strokes. As in [Kov04a] the orientation of the placed strokes will be determined from the nearest weighted edge direction. Thus main contours are preserved and artifacts caused by badly placed strokes are circumvented. In the following the focus map extraction and the segmentation steps are described, and then details of the painting process will follow.

Focus Extraction

For automatically extracting the relevant regions of the model images we use the properties of localized blurring functions obtained with a method based on blind deconvolution [Ric72a].

We look at the model image as an original image with some distortion (g model image, f original, h distortion)

$$g = f * h$$

and run a few iterations of localized blind deconvolution [Kov05a] for obtaining an estimation of the local variation in focus/blur over the areas of the model image

$$h_{k+1}(r) = \frac{h_k(r)}{\gamma}\left[f_k(r) * \frac{g}{g_k} \right]$$

where $g_k = f_k * h_k$ the current local reconstruction and γ is a local normalization constant dependent on the size of the local region of support and the estimated blur function.

Then we use the differences in local blur function estimation to classify the image areas relative to each other based on the estimated local focus. The main idea of this approach is that local blurring function estimates will vary according to the local blurriness/focus-ness of the image. For an example see Fig. 1.

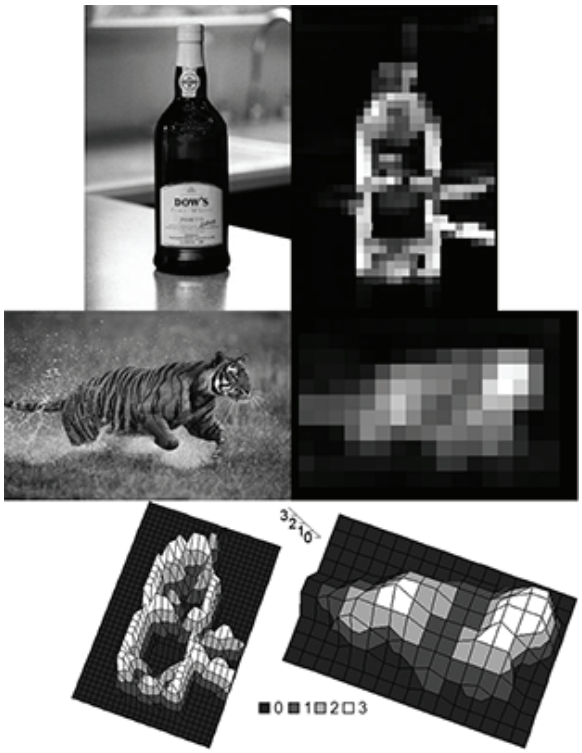


Figure 1. Samples for relative focus map extraction. Top: inputs and relative focus maps, bottom: relative focus surfaces with four classes.

Color Segmentation

Our goal with color segmentation is twofold. First, it provides us the a regional segmentation map based on color similarity and at the same time a means to quickly generate low-detail background.. Secondly, to prepare the grounds for later cartoon-style 2D stroke-based rendering. Our approach is based on the hue-intensity based segmentation approach in [Luc01a, Zha00a]. The main steps of the color segmentation are:

1. Convert the model image's color space from RGB to HSV (hue, saturation, value) and normalize the obtained values for the next step:

$$\begin{cases} h = 255(h - h_{\min})/(h_{\max} - h_{\min}) \\ s = 255s/s_{\max} \\ v = 255v/v_{\max} \end{cases}$$

2. Perform anisotropic diffusion on the hue and on the value planes [Per90a] to blur the image by space-variant blur with low effect on contours.
3. Perform K-means clustering [Mac67a] on the combined hue+value color planes into 8-32 classes.
4. Extract and assign colors to the classes of the clustering: the most frequent color over the whole extracted class area. This is done by calculating color histograms of the respective cluster, pick the most frequent color and assign it to the pixels belonging to the cluster.

For an example see Fig. 2.

Figure 2. Samples for color segmentation and cluster colorization. Left column: input, middle: segmentation into 32 classes, right: colorized clusters.

Painting

The rendering follows the steps outlined earlier. The painting process itself is done by placing strokes on the canvas, each stroke having parameters: type (template identification), size, position, orientation (determined from the extracted edge directions) and color (for each stroke the most frequent color under the stroke area). The process is controlled by the relative change in error caused by the placed strokes between the current state of the painting on the canvas and the previous state. States are checked after every thousand placed strokes. If the relative error change between two consecutive states is below a threshold (ε={5,0.7,0.002} for the respective stroke scales) we switch for the next layer.

Thus after the focus map extraction the low-detail layer is generated. This phase can be achieved by two means: either by covering with larger scaled strokes, or by using the color map obtained by the color segmentation process described above. In either case, after the generated background the relevant regions are refined by the smallest stroke scale. This way the following outputs can be generated:

- painted on painted: background and extracted regions both are rendered;
- painted on original: painted extracted regions are drawn upon the original model;
- original on painted: the model image contents over the extracted regions are drawn upon the rendered background.

From the algorithm's point of view it does not matter what kind of artificial strokes are used during the painting process. In the present painting simple rectangular stroke templates are used. Fig. 3. contains an example for these variations. Fig. 4. shows example rendered outputs using the technique outlined in this paper (for the painted on painted case).

The above presented method builds the painted image by generating a coarser layer and a finer layer on top of it, by using three scales of strokes. Of course, the use of more layer scales could be possible, if more refined transition between the background and the fine layer is desired. But in this method our goal was exactly the production of a finer layer on a coarser one, thus achieving a more cartoonish feel.

Figure 3. Painted outputs. Top left: input, and focus map, top right: painted object on painted background, bottom left: painted on original, bottom right: original on painted.

3. CONCLUSION

In this paper we presented an automatic approach for region of relevance based multilayer 2D painterly rendering, by application of a blind deconvolution based local relative focus difference classification approach. We also introduced an extension with color segmentation for fast low-detail background generation for the above technique. The resulting approach is able to generate pleasant images with a painterly look and feel, with detail-variation relative to local focus-based relevance.

4. ACKNOWLEDGMENTS

This work has been supported by the Hungarian Research Fund (OTKA T049001).

Figure 4. Rendering samples. Top: input images and their focus maps. Middle row: painted objects on low-detail painted background. Bottom row: painted objects on segmentation-colorized background.

5. REFERENCES

[Dec02a] DeCarlo, D., and Santella, A. Stylization and Abstraction of Photographs. Proceedings of ACM SIGGRAPH 2002, p. 769-776, 2002.

[Goo02a] Gooch, B., Coombe, G., and Shirley, P. Artistic Vision: Painterly Rendering Using Computer Vision Techniques. Proceedings of NPAR, p. 83-90, 2002.

[Hae90a] Haeberli, P. Paint By Numbers: Abstract Image Representation. Computer Graphics, 24, 4, Aug. 1990. p. 207-214.

[Her98a] Hertzmann, A. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. Proceedings of ACM SIGGRAPH 98, p. 453-460, 1998.

[Kov04a] Kovács, L., and Szirányi, T. Painterly Rendering Controlled by Multiscale Image Features. Proceedings of SCCG. p. 183-190, 2004.

[Kov04b] Kovács, L., and Szirányi, T. Efficient Coding of Stroke-Rendered Paintings. Proceedings of the 17th ICPR, IAPR&IEEE, p. 835-839, 2004.

[Kov05a] Kovács, L., and Szirányi, T. Relative Focus Map Estimation Using Blind Deconvolution, Optics Letters, 30, p. 3021-3023, 2005.

[Lin98a] Lindeberg, T. Edge Detection and Ridge Detection With Automatic Scale Selection. International Journal of Computer Vision, 30, 2, p. 117-154, 1998.

[Lit97a] Litwinowicz, P. Processing Images and Video for An Impressionist Eect. Proceedings of ACM SIGGRAPH 97, p. 407-414, 1997.

[Luc01a] Lucchese, L., and Mitra, S. K. Color Image Segmentation Through Independent Anisotropic Diffusion of Complex Chromaticity and Lightness. Proceedings of ICIP, vol. 1, p. 746-749, 2001.

[Mac67a] MacQueen, J. B. Some Methods For Classification And Analysis of Multivariate Observations. Proceedings of the 5th Berkeley Symposium on Mathematical statistics and Probability, vol.1, p. 281-297, 1967.

[Par04a] Park, Y., and Yoon, K. Adaptive Brush Stroke Generation for Painterly Rendering. Eurographics 2004 Short Presentations, 2004.

[Per90a] Perona, P. and Malik, J. Scale-Space and Edge Detection Using Anisotropic Diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12, 7, p. 629-639, 1990.

[Ric72a] Richardson, W.H. Bayesian-Based Iterative Method of Image Restoration. JOSA 62 55–59 (1972)

[San02a] Santella, A., and DeCarlo, D. Abstracted Painterly Rendering Using Eye-Tracking Data. Proceedings of NPAR, p. 75-82, 2002.

[Szi00a] Szirányi, T., and Tóth, Z. Random Paintbrush Transformation. Proceedings of the 15th ICPR, IAPR&IEEE, p. 155-158, 2000.

[Zha00a] Zhang, C., and Wang, P. A New Method of Color Image Segmentation Based on Intensity and Hue Clustering. Proceedings of ICPR, vol. 3, p. 3617-3620, 2000.

LOOKING THROUGH THE EYES OF THE PAINTER: FROM VISUAL PERCEPTION TO NON-PHOTOREALISTIC RENDERING

Roberto Lam
University of Algarve
Escola Superior de Tecnologia
Campus da Penha
8005-139, Faro, Portugal
rlam@ualg.pt

João Rodrigues
University of Algarve
Escola Superior de Tecnologia
Campus da Penha
8005-139, Faro, Portugal
jrodrig@ualg.pt

J.M.H. du Buf
University of Algarve
Vision Laboratory - FCT
Campus de Gambelas
8005-072, Faro, Portugal
dubuf@ualg.pt

ABSTRACT

In this paper we present a brief overview of the processing in the primary visual cortex, the multi-scale line/edge and keypoint representations, and a model of brightness perception. This model, which is being extended from 1D to 2D, is based on a symbolic line and edge interpretation: lines are represented by scaled Gaussians and edges by scaled, Gaussian-windowed error functions. We show that this model, in combination with standard techniques from graphics, provides a very fertile basis for non-photorealistic image rendering.

Keywords

Perception, models, multi-scale representations, brightness, focus-of-attention, non-photorealistic rendering.

1. INTRODUCTION

Painters have learned to observe, to select relevant information, and to translate this information into a representation that they want to transfer, in some preferred style like pointillism (e.g. Seurat) or impressionism (e.g. van Gogh). These two styles can be seen as extreme examples of Level-of-Detail (LoD), which is strongly related to the processing in the retina and visual cortex. In addition, painters can apply coarse-to-fine scale and background-to-foreground "rendering," plus special techniques like clair-obscur and wet-in-wet, using a broad array of brushes and palette knives. Available pigments allow to approximate any real color, although many painters prefer a very limited color gamut.

Trying to understand painters, their techniques, and visual aesthetics in general, is a challenge, especially when referring to physical processes in the eyes and brain [Zeki, 2000; Livingstone, 2000]. An even bigger challenge is to exploit these processes in trying to simulate techniques and styles of certain painters. This requires state-of-the-art models of image representations in the visual cortex together with insight into higher-level cognitive effects. For example, *symbolic pointillism* [Krüger and Wörgötter, 2003] renders small circles at edge positions by exploiting principles of Gestalt theory, such as good continuity grouping, in order to present meaningful image information.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

Visual perception seems to be an effortless, transparent process. In reality it is the result of a combination of many different, complicated and still partly understood mechanisms in the retinas, visual cortex and higher brain areas. Hypercolumns in cortical area V1 contain a stack of scale- and orientation-tuned cells that provide retinotopic (neighborhood preserving) image-representation and feature maps: even and odd simple cells, complex and end-stopped cells, plus many grouping cells that serve to detect basic features like lines and edges, bars and gratings, keypoints and saliency maps for Focus-of-Attention (FoA), motion and disparity. This information is relayed to higher brain areas, via ventral and dorsal data streams, in combination with top-down data streams, forming the what and where subsystems [Deco and Rolls, 2004; Rensink, 2000].

Receptive fields of even and odd simple cells, which can be seen as anisotropic quadrature filters, are commonly modeled by complex Gabor functions with real (even) and imaginary (odd) parts. Phase invariant complex cells are then modeled by

combining activities of even and odd simple cells: the modulus of the complex Gabor response (see Section 2 for definitions). Recently, models of other cortical cells have been developed, for example bar and grating cells [Petkov and Kruizinga, 1997; Santos and du Buf, 2001] and end-stopped cells [Heitger et al.,1992; Rodrigues and du Buf, 2004], the first detecting isolated bars or periodic patterns, the latter junctions and points of high curvature (keypoints).

In this paper we illustrate part of the visual representation in the visual cortex, i.e. the multi-scale line/edge coding necessary for developing a 2D model of brightness perception, and the multi-scale keypoint (vertex) representation that can be used for constructing a saliency map for modeling Focus-of-Attention (FoA). Together, these representations can be used for non-photorealistic rendering (NPR), i.e. new NPR schemes completely based on models of perception: painterly rendering by using brushes of different sizes [Hertzmann, 1998] and image stylization [DeCarlo and Santella, 2002] with LoD controlled by FoA.

2. CELL MODELS

Gabor quadrature filters provide a model of cortical simple cells. In the spatial domain (x, y) they consist of a real cosine and an imaginary sine, both with a Gaussian envelope. A receptive field (RF) is denoted by (see e.g. [Rodrigues and du Buf, 2005a]):

$$g_{\lambda,\sigma,\theta,\varphi}(x,y) = \exp\left(-\frac{\tilde{x}^2+\gamma\tilde{y}^2}{2\sigma^2}\right)\cos\left(2\pi\frac{\tilde{x}}{\lambda}+\varphi\right),$$

$\tilde{x} = x\cos\theta + y\sin\theta; \tilde{y} = y\cos\theta - x\sin\theta,$ where the aspect ratio γ = 0.5 and σ determines the size of the RF. The spatial frequency is 1/λ, λ being the wavelength. For the bandwidth σ/λ we use 0.56, which yields a half-response width of one octave. The angle θ determines the orientation (we use 8 orientations), and φ the symmetry (0 or π/2). We apply a linear scaling between $f_{\min}$ and $f_{\max}$ with a few discrete scales, or hundreds of contiguous scales.

The responses of even and odd simple cells, which correspond to the real and imaginary parts of a Gabor filter, are obtained by convolving the input image with the RF, and are denoted by $R^E_{s,i}(x,y)$ and $R^O_{s,i}(x,y)$, s being the scale and i the orientation $(\theta_i = i\pi/(N_\theta - 1))$ and N_θ the number of orientations. In order to simplify the notation, and because the same processing is done at all scales, we drop the subscript s. The responses of complex cells are modelled by the modulus

$$C_i(x,y) = \left[\left\{R_i^E(x,y)\right\}^2 + \left\{R_i^O(x,y)\right\}^2\right]^{1/2}.$$

There are two types of end-stopped cells [Heitger et al., 1992], i.e. single (S) and double (D). If $[.]^+$ denotes the suppression of negative values, and $\hat{C}_i = \cos\theta_i$ and $\hat{S}_i = \sin\theta_i$, then

$$S_i(x,y) = \left[C_i\left(x + d\hat{S}_i, y - d\hat{C}_i\right) - C_i\left(x - d\hat{S}_i, y + d\hat{C}_i\right)\right]^+;$$

$$D_i(x,y) = \left[C_i(x,y) - \tfrac{1}{2}C_i\left(x + 2d\hat{S}_i, y - 2d\hat{C}_i\right) - \tfrac{1}{2}C_i(x - 2d\hat{S}_i, y + 2d\hat{C}_i)\right]^+.$$

The distance d is scaled linearly with the filter scale s, i.e. d = 0.6s. All end-stopped responses along straight lines and edges need to be suppressed, for which we use tangential (T) and radial (R) inhibition:

$$I^T(x,y) = \sum_{i=0}^{2N_\theta-1}\left[-C_{i \bmod N_\theta}(x,y) + C_{i \bmod N_\theta}\left(x + d\hat{C}_i, y + d\hat{S}_i\right)\right]^+;$$

$$I^R(x,y) = \sum_{i=0}^{2N_\theta-1}\left[C_{i \bmod N_\theta}(x,y) - 4.C_{(i+N_\theta/2) \bmod N_\theta}\left(x + \frac{d}{2}\hat{C}_i, y + \frac{d}{2}\hat{S}_i\right)\right]^+,$$

where $(i + N_\theta/2) \bmod N_\theta \perp i \bmod N_\theta$.

All responses of the end-stopped cells $S(x,y) = \sum_{i=0}^{N_{\theta-1}} S_i(x,y)$ and $D(x,y) = \sum_{i=0}^{N_{\theta-1}} D_i(x,y)$ are inhibited by $I = I^T + I^R$ for obtaining the keypoint maps $K^S(x,y) = S(x,y) - gI(x,y)$ and $K^D(x,y) = D(x,y) - gI(x,y)$, with $g \approx 1.0$, and then the final keypoint map $K(x,y) = \max\left\{K^S(x,y), K^D(x,y)\right\}$. In the multi-scale case keypoints are detected the same way as done above, but now by using $K_s^S(x,y) = S_s(x,y) - gI_s(x,y)$ and $K_s^D(x,y) = D_s(x,y) - gI_s(x,y)$. For more details and results see [Rodrigues and du Buf, 2005a].

Figure 6 shows the "tree" image which will be used as reference. Figure 1 shows at the left the finest scale (λ = 4), and at the right a coarser scale (λ = 16), from top to bottom: responses of complex cells, combined even and odd simple cells, plus single and double end-stopped cells. Complex and simple cell

activities are only shown for the local dominant orientation, i.e. this is a selection of all cell activities.

We can see that the basic cell responses are very fuzzy, without postprocessing even completely useless. Figure 2 (upper part) shows four event maps at the same two scales $\lambda = 4$ (top) and $\lambda = 16$ (bottom). Despite the fuzzy responses of end-stopped cells (Fig. 1), the inhibition process leads to precise detection of keypoints, even at coarse scales. Figure 2 (left) shows line/edge maps, i.e. the positions of detected lines and edges and the event type, for example positive line or negative edge, coded by gray level and superimposed on the input image. Not shown in Fig. 2 are line/edge orientation and amplitude.

A detailed description of line and edge detection is beyond the scope of this paper, hence we refer to [van Deemter and du Buf, 1996)] and [Rodrigues and du Buf, 2004]. The basic idea is the following: a local maximum of responses of complex cells gives a first, but normally inaccurate estimate of the position. The near zero-crossing of even or odd simple cells is much more precise, and the combination of even and odd simple cells' local extremum (max or min) and zero-crossing determines what there is, for example a negative line or a positive edge [du Buf, 1993].

The four maps shown in Fig. 2 represent event cells, i.e. cells which only respond when at their retinotopic position there is a line, an edge or a vertex in the visual input. Line/edge orientation and scale are coded by different cells, and the activity of each cell reflects the amplitude of the underlying line or edge through the response of the corresponding complex cell. This symbolic representation or neural code is used at higher levels in the visual cortex, for brightness perception (see next section) and object detection and recognition.

Keypoint cells are binary, i.e. they respond or they don't. An analysis over many scales showed that keypoints are stable at important structures, i.e. local image complexity, that the stable scale intervals indicate the scale of the structures, and that they can be used for object and face detection [Rodrigues and du Buf, 2005a; Rodrigues and du Buf, 2005b]. In addition, since local image complexity provides very important information for planning fixation points and saccades of the eyes, keypoints are thought to play a significant role in Focus-of-Attention by means of a saliency map [Rodrigues and du Buf, 2005a].

If we assume that retinotopic projection is maintained throughout the visual cortex, the activities of all keypoint cells at the same position (x,y) can be easily summed over scale s, which leads

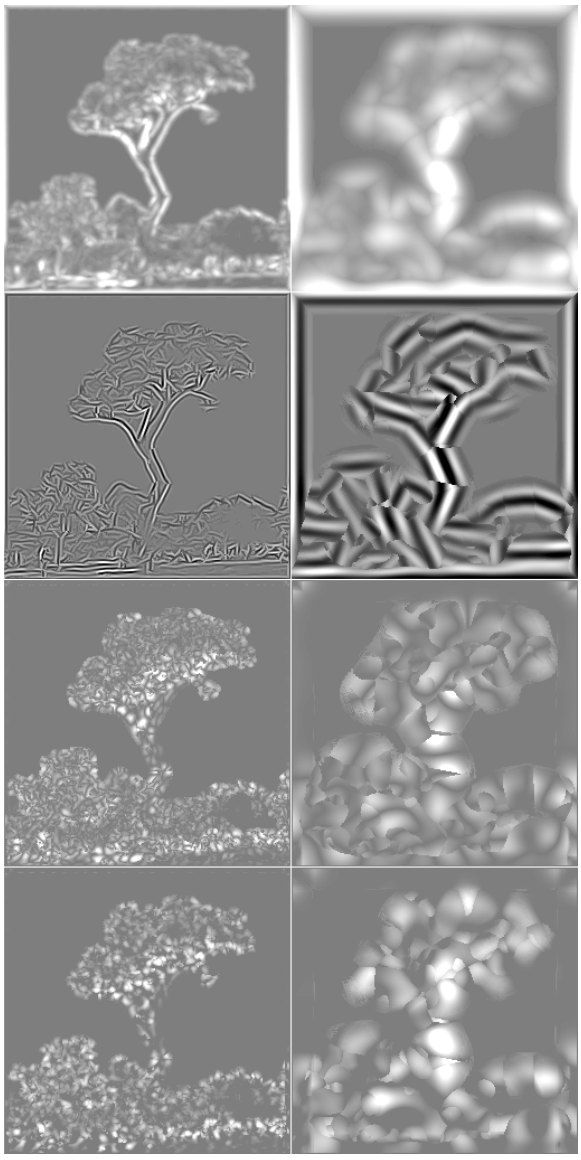


Figure 1. Top to bottom: responses of complex, simple, and single and double end-stopped cells at two scales.

to a very compact, single-layer map. At the positions where keypoints are stable over many scales, this summation map will show distinct peaks at centers of objects, important sub-structures and contour landmarks. The height of the peaks can provide information about the relative importance. In addition, this summation map, with some simple processing of the projected trajectories of unstable keypoints, like a dynamic lowpass filtering related to the scale and non-maximum suppression [Rodrigues and du Buf, 2005a], might solve the segmentation problem: the object center is linked to important sub-structures, and these are linked to contour landmarks. Such a mapping or data stream is data-driven and bottom-up, and could be combined with top-down processing from inferior temporal cortex (IT) in

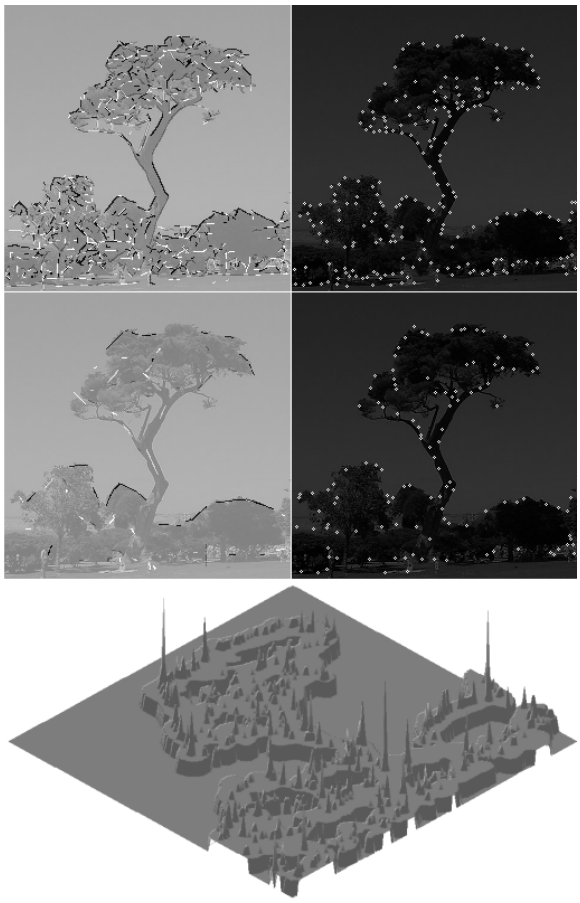


Figure 2. Top: event maps with detected lines and edges (left) and keypoints (right) at two scales. Bottom: projected saliency map.

order to actively probe the presence of certain objects in the visual field [Deco and Rolls, 2004]. In addition, the summation map with links between the peaks might be available at higher brain areas where serial processing occurs for e.g. visual search.

Apart from detected keypoints at two scales, Fig. 2 also shows a projected saliency map obtained over a big scale interval ($\lambda = [4, 32]$; $\Delta\lambda = 1$). This map contains distinct peaks at positions where keypoints are stable over scale intervals, and the lowpass filtering coupled to the scale of driving complex cells leads to regions which connect the peaks. It has been shown that such a saliency map corresponds to the map of fixation points as measured by eye tracking during inspection of faces [Rodrigues and du Buf, 2005b].

3. MULTI-SCALE LINE/EDGE REPRESENTATION AND BRIGHTNESS

There are very few models for explaining brightness perception and illusions like Mach bands [Pessoa, 1996]. One brightness model [du Buf, 1994; du Buf and Fischer, 1995] assumes that lines and edges are

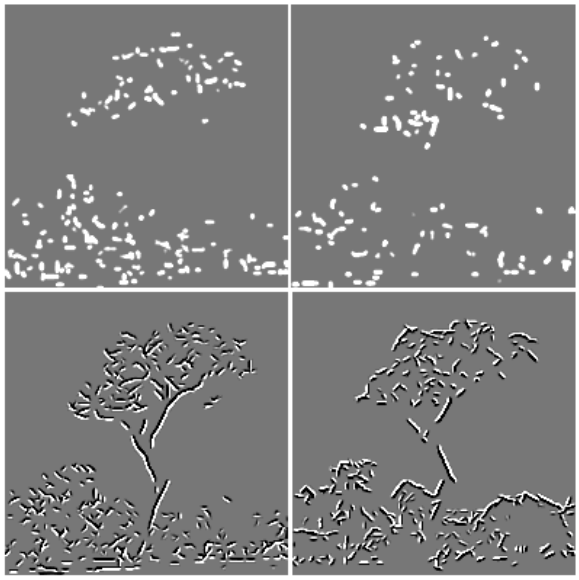


Figure 3. Symbolic brightness interpretation of positive and negative lines (top row) and positive and negative edges (bottom row) at the finest scale.

detected at all possible scales, and that these are interpreted symbolically: a responding "line cell" implies that, at the cell's retinotopic position, there is a line with a certain orientation, amplitude and scale, i.e. a Gaussian profile with a size that depends on the scale of the underlying simple and complex cells. The same happens in the case of an "edge cell," but with a bipolar Gaussian-windowed error-function profile (see Fig. 3). It is important to stress that the interpretation of all responding line and edge cells leads to a *virtual* representation of the input image, i.e. there is no cell layer in which all the information is summed in order to create a retinotopic brightness map[1]. The fact that responses of simple cells are the same in the case of lines and ramp edges leads to an elegant explanation of Mach bands [du Buf, 1994], and an extended model was shown to explain also brightness induction (simultaneous contrast and assimilation) and other illusions [du Buf and Fischer, 1995]. This model is being extended from 1D to 2D, which involves complicating factors like the size of receptive fields in the case of curved lines and edges, see [van Deemter and du Buf, 1996]. In future models it may be possible to exploit end-stopped and keypoint cells in order to model transparency.

Figure 4 shows two opposite brightness induction effects and preliminary model predictions. In simultaneous brightness contrast, the circles, which are physically the same under homogeneous

[1] This would require yet another "observer" inside our brain.

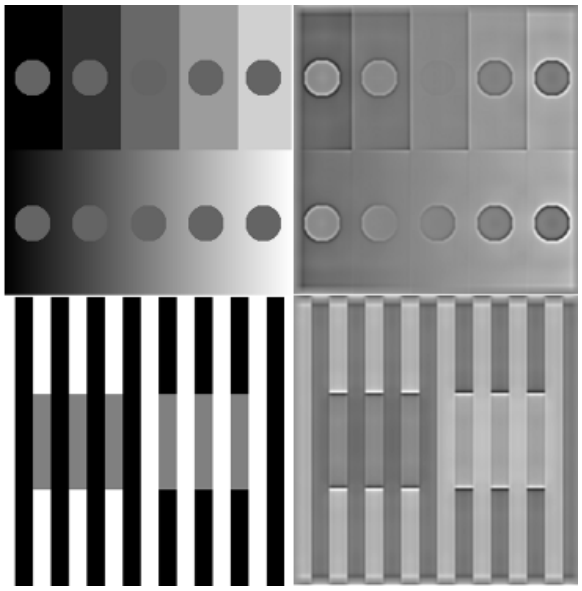


Figure 4. Brightness illusions. Top-left: simultaneous brightness contrast. Bottom-left: assimilation. All circles (top) and gray bars (bottom) have the same reflectance but appear different. Right: model predictions.

illumination, appear different because the background pushes brightness in the opposite direction, i.e. the circles to the left appear brighter and the ones to the right darker. In assimilation, the flanking black and white bars pull the brightness of the gray bars, which also are physically equal, in the same direction: the left bars appear darker than the right bars.

Model predictions shown in Fig. 4, which agree with our brightness impression, are based on mixing one lowpass-filtered image providing a global background, with weighted contributions from the symbolic line and edge representations. This is shown in Fig. 5, which illustrates the application of visual reconstruction to NPR. The top image is obtained by summing a lowpass-filtered version of the tree image and detected lines and edges at four scales: event positions (Fig. 2) and line/edge profiles (Fig. 3). Comparing Fig. 6 (top) with the reconstruction (top Fig. 5) we can see that the latter is not perfect. This due to the fact that only four scales have been used.

4. NON-PHOTOREALISTIC RENDERING

A perfect reconstruction is not a goal in NPR. The idea is to create an image that looks like an input image, but with an artistic twist. For example, Hertzmann (1998) simulated painting with (spline) brush strokes, the sizes of the brushes being linked to the sizes of Gaussian kernels used in lowpass

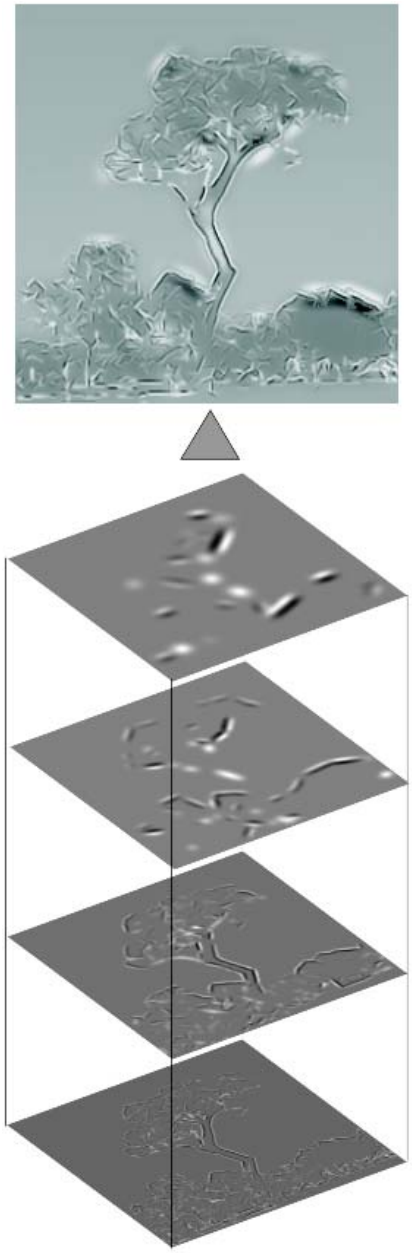


Figure 5. Multi-scale NPR of the tree image based on one lowpass filter and four line/edge scales. The bottom image is the sum of the four images shown in Fig. 3.

filtering the input image. Our brightness model on the basis of symbolic line/edge representation provides a perceptually justified alternative: a line is a sequence of positions with intensities, the profile (scale) being truncated by a Gaussian function. In other words, the profile can be seen as a brush and the sequence of positions as a stroke. In addition, rendering can be combined with image stylization [DeCarlo and Santella, 2002] in which our model of FoA, a saliency map as shown in Fig. 2, can be used

to select brush strokes (line/edge scales) dynamically: small brushes only at positions with peaks caused by local image complexity. The latter

Figure 6. Simulated oil canvas.

idea has not yet been implemented (see also Discussion). Here we show a few results obtained with the method applied to reconstruct the tree image (Fig. 5), i.e. lowpass filtering plus four line/edge scales.

Figure 6 (top) shows the tree image in color, and a simulated oil canvas produced with Linux GNU tool GIMP. The images used for rendering this scene are shown in Fig. 5. Because our models are restricted to grayscale (color perception may be added later), we picked a few colors of the input image to create RGB lookup tables and color gradients for the rendering layers. The result was obtained with no manual editing. The canvas texture is a basic GIMP feature. As can be seen, the result is an impressionist view of the tree image, in which global structures (trees, lawn) are preserved but not in detail.

Figure 7. Simulated watercolor.

Figure 8. Simulated crayon.

Figure 7 shows a picture taken across a Venice (Italy) canal. Instead of an oil canvas, we tried to obtain a watercolor rendering of type "Turner in Venice." Like Turner, we used very unsaturated and diffuse colors, and the contrast of the input lines and edges was reduced. In contrast with the rendering of the tree image, manual editing was necessary in order to approximate Turner's style in the sky and water. A more realistic rendering of Turner's textures (windows, facades) is probably only possible by substituting the line and edge representation by real, discrete brush strokes in combination with wet-in-wet painting. However, it should be stressed that the NPR is entirely based on the information present in the input image. We cannot (yet) simulate the hand of the painter while painting fine, repetitive structures and textures.

Figure 8 shows the Fiona image with NPR. No manual editing was applied, and the typical crayon effect was added by Corel's Photo-Paint standard "impressionist" filter.

Figure 9 shows a first result obtained with painterly rendering and discrete brush strokes. The image was created in two steps. First, a background was created on the basis of lowpass-filtered RGB components. Big, elliptical brush strokes were applied at random positions and with random orientations. Three colors were picked, at the center of the stroke and at the two end points. If the color at one end point deviated significantly from the average of the other two colors, the brush stroke was not applied. This prevents from introducing wrong colors in areas where no line/edge information is available, for

example in the sky close to the tree. This process was repeated until the entire background image was covered. The brush strokes cannot be seen in the sky,

Figure 9. Painterly rendering.

because of the very small gradient. Second, line and edge representations (Fig. 3) were converted into discrete brush strokes, by splitting long lists of (x,y) positions into smaller lists, such that the aspect ratio of the brush strokes was about the same at different scales (brush sizes). Lines were rendered with single-color strokes, whereas edges were rendered with two, parallel strokes. The color of each stroke was picked in the unfiltered input image, at the center of the stroke. This process was applied from coarse to fine scales, each stroke substituting previously rendered colors (painting wet-on-dry). All brush strokes, including those for creating the background, were modeled by triangle lists in combination with opacity in the alpha channel, for obtaining a spray-like effect, using standard features of OpenGL. Figure 9 shows that the final result approaches the effect of a real painting, for example in the tree top. In this first result we already applied color equalization by the ACE model, but not yet saliency maps for stylization (see below).

5. DISCUSSION

A set of Gabor filters with suitably chosen characteristics in combination with one lowpass filter, such that the sum of all transfer functions is about constant over the entire frequency domain, can be seen as a linear allpass filter and allows to reconstruct the input image. This idea is exploited in image coding by (Gabor) wavelet transforms. Our visual system does not reconstruct the input image in the same way, i.e. by summing responses of simple cells. Our visual system does not even construct a virtual representation of the entire visual field, see change blindness and the limited capacity of the attentional system [Rensink, 2000]. The what and where systems serve object detection and are based on feature-extraction engines in area V1 and higher areas.

In this paper we illustrated a few, basic processing steps and cell models in V1. We also showed that one brightness model that is based on a symbolic line and edge representation can be used for non-photorealistic rendering with many possibilities. However, our goal is photorealistic rendering in the sense of developing a 2D or even 3D brightness model that is capable of predicting psychophysical data and visual illusions like Mach bands and brightness induction (simultaneous contrast and assimilation). Figure 5 shows the tree image (re)constructed by assuming a lowpass filter in combination with a multiscale summation of line and edge representations, but without using nonlinear amplitude transfer functions for the different scales associated with nonlinear brightness perception [du Buf and Fischer, 1995].

The NPRs of the tree image (Figs 6, 9), as for the simulated watercolor and crayon (Figs 7, 8), are preliminary results to show some possibilities of our approach, i.e. they are *probationes pennae* or pen tests. The next step will be to optimize the rendering process such that results cannot be distinguished from real paintings, using real brush strokes in the line of Hertzmann's (1998) painterly rendering. Long brush strokes can be modeled by splines, and these can be modified by splitting them into short, randomized strokes, or they can be linearized, in order to create impressionist or expressionist renderings. Gestalt theory, as applied in *symbolic pointillism* [Krüger and Wörgötter, 2003], can be used to retain crucial image information. This idea is strongly linked to image stylization and abstraction [DeCarlo and Santella, 2002], but instead of recording eye movements it will be possible to exploit our keypoint-based saliency map to control the Level-of-Detail by selecting brush sizes on the basis of local image complexity. Few but vivid colors are often used in cartoon- or comics-like renderings, such as those shown by DeCarlo and Santella, and a recent model of color constancy called ACE (Automatic Color Equalization) [Rizzi et al., 2003] can be used to convert dull colors into vivid ones. There are numerous ideas for creating NPRs [Sousa, 2003].

Non-photorealistic rendering is a spin-off and can, preferably in collaboration with artists and specialists of color perception, be used to develop new “filters” for tools like GIMP and PaintShop. However, instead of applying a certain effect at all image positions,

like GIMP's randomized brush strokes that can be influenced by image intensity or the canvas texture used in the tree image, more intelligent filters can be developed. For example, developing a "Seurat filter" might be relatively straightforward because of the fine brush strokes ("sampling"), but a "van Gogh filter" requires, apart from knowledge about brushes and colors, deep insight into his cognitive perception [Zeki, 2000; Livingstone, 2000]. Nevertheless, our method may provide the basis for a systematic study of the emotional impact of different rendering techniques [Halper et al., 2003; Healey and Enns, 2002] and the importance of aspects like 3D cues, color and texture [Kjelldahl, 2003]. In addition, instead of applying NPR to color images, it is possible to explore other data sets, with the goal of highlighting specific data structures: layers of multidimensional data sets can be visualized by means of perceptually based brush strokes [Healey et al., 2004].

6. ACKNOWLEDGMENTS

This research is partly financed by PRODEP III Medida 5, Action 5.3, and by the FCT program POSI, framework QCA III.

7. REFERENCES

DeCarlo, D. and Santella, A. (2002) "Stylization and Abstraction of Photographs". Proc. SIGGRAPH 2002, pp. 769-776.

Deco, G. and Rolls, E. (2004) "A neurodynamical cortical model of visual attention and invariant object recognition".Vision Res., (44):621–642.

du Buf, J. (1993) "Responses of simple cells: events, interferences and ambiguities". Biol. Cybern., 68:321–333.

du Buf, J. (1994) "Ramp edges, Mach bands, and the functional significance of the simple cell assembly". Biol. Cybern.,70:449–461.

du Buf, J. and Fischer, S. (1995) "Modeling brightness perception and syntactical image coding". Optical Eng. 34(7):1900–1911.

Halper, N. et al. (2003) "Towards an understanding of the psychology of non-photorealistic rendering". Comp. Visualistics, Media Informatics, and Virtual Communities, Vol. 11, pp. 67–78.

Healey, C. and Enns, J. (2002) "Perception and painting: A search for effective, engaging visualizations". IEEE Comp. Gr. Appl., 22(2):10–15.

Healey, C. et al. (2004) "Perceptually based brush strokes for non-photorealistic visualization". ACM Trans. Graphics, 23:64–96.

Heitger, F et al. (1992) "Simulation of neural contour mechanisms: from simple to end-stopped cells". Vision Res., 32(5):963–981.

Hertzmann, A. (1998) Painterly rendering with curved brush strokes of multiple sizes. Comp. Graphics Proc. (SIGGRAPH 98), pp. 453–460.

Kjelldahl, L. (2003) "A survey of some perceptual features for computer graphics and visualization". Annual SIGRAD Conf., Umea Univ., Sweden, Linkoping Conf. Proc. ISSN 1650-3686.

Krüger, N. and Wörgötter, F. (2003) "Symbolic pointillism: Computer art motivated by human perception". Proc. Symp. Artif. Intell. Creativity in Arts and Science – AISB 2003, pp. 36–40.

Livingstone, M. (2000) "Vision and art: the biology of seeing". Abrams, New York (NY).

Pessoa, L. (1996) "Mach bands: how many models are possible? Recent experimental findings and modeling attemps". Vision Res., 36:3205–3227.

Petkov, N. and Kruizinga, P. (1997) Computational models of visual neurons specialised in detection of periodic and aperiodic visual stimuli". Biol. Cybern., 76:83–96.

Rensink, R. (2000) "The dynamic representation of scenes". Visual Cognition, 7(1-3):17–42.

Rizzi, A., Gatta, C. and Marini, D. (2003) A new algorithm for unsupervised global and local color correction. Patt. Recogn. Lett., 24:1663-1677.

Rodrigues, J. and du Buf, J. (2004) "Visual cortex frontend: integrating lines, edges, keypoints and disparity". Proc. Int. Conf. Image Anal. Recogn., Springer LNCS 3211, pp. 664–671.

Rodrigues, J. and du Buf, J. (2005a) "Multi-scale cortical keypoint representation for attention and object detection". 2nd Iberian Conf. Patt. Recogn. Im. Anal., Springer LNCS 3523, 255-262.

Rodrigues, J. and du Buf, J. (2005b) "Multi-scale keypoints in V1 and face detection". 1st Int. Symp. Brain, Vision and Artif. Intell., Springer LNCS 3704, 205-214.

Santos, L. and du Buf, J. (2001) "Improved grating and bar cell models". 7th Neural Computing in Psychology Worksh., Brighton, Sept. 17-19.

Sousa, M. Theory and practice of non-photorealistic graphics: Algorithms, methods, and production systems. Course Notes for SIGGRAPH 2003 http://pages.cpsc.ucalgary.ca/˜mario/.

van Deemter, J. and du Buf, J. (1996) "Simultaneous detection of lines and edges using compound Gabor filters". Int. J. Patt. Recogn. and Artif. Intell., 14(6):757–777.

Zeki, S. (2000) "Inner vision: an exploration of art and the brain". Oxford Univ. Press.

Comics-Like Motion Depiction from Stereo

Danijela Marković and Margrit Gelautz
Institute for Software Technology and Interactive Systems, Vienna University of Technology
Favoritenstrasse 9-11/188/2, A-1040 Vienna, Austria
e-mail: {markovic, gelautz}@ims.tuwien.ac.at

ABSTRACT

In this paper, we present an algorithm to depict motion in comics-like form, with an artistic drawn-like representation of the scene, from a stereo image sequence. The input to the algorithm is a natural scene, along with a user-defined set of parameters that define the tone and stylistic properties of the image to be produced. The algorithm uses a dense disparity map, computed from the input stereo video, to preserve the perspective perception of the stylized image by drawing each stroke in a direction determined by the stereo derived disparity layers. To outline important features in the image, we utilize the contour edges provided by the Edge Combination algorithm. In the next step, we detect motion of the objects in the scene by tracking points that are close to the dominant edges of the Edge Combination image. The extracted dominant structure is further used to obtain a larger variety of styles for visualizing the motion trajectories. The output of the algorithm is a drawn-like form of the original scene with the motion highlighted in imitation of comics produced by hand.

Keywords: Real scene, stereo, Edge Combination, image-based rendering, motion depiction, tracking.

1 INTRODUCTION

The heart of comics lies in the space between the panels where the reader's imagination makes still pictures come alive! [McC00]

Although there is an ongoing discussion about the historical roots of comics and their artistic contribution, reputed to be the first comic strip magazine is "Ally Sloper's Half Holiday" published in 1884. It was one of the most famous and most popular of all Victorian comics, featuring a regular character, Ally Sloper. However, the great boom that prompted the beginning of comics as an ongoing popular art form was Richard Fenton Outcault's cartoon series "Hogan's Alley" for Joseph Pulitzer's New York World in 1895. The Yellow Kid, as he later came to be known for his yellow nightshirt, became the first comic character to serve as a marketing tool for the sale of newspapers. This set the bases for a new kind of art with the adventures of superheroes and monsters in stories written in a powerful language of visual symbols. [Sab01, Har99]

Comic books lost a lot of their popularity with the development and expansion of 3D animation and new media. Today a great variety of commercial applications give to the user the ability to create interesting artwork even without the requirement of good drawing skills. Contrary to this, comic books are usually created by skilled and creative artists who use a style of drawing unique for this type of art.

Comic books can be compared to silent movies, having dialogs in a written form and telling a story through sequences of images with a very important difference: comics use one image to present a sequence of frames. The form to visually depict the motion in just one image has been a big challenge for artists through history and resulted in different methods like: dynamic balance, multiple stroboscopic images, affine shear, photographic blur, and action lines [Cut02]. Today, pictorial description of the motion is particularly interesting for image stylization and initiated the development of a number of systems [MSS99, BE01, CRH03, KE05].

In this work we concentrate on a comic books' style of motion expression, where few lines suggest an action in the scene through motion lines and multiple contours. Similar to our work, Masuch et al. [MSS99] depict motion, but different to our system they use a 3D scene as input, having access to the 3D geometry and precise motion information of objects. The overall goal of our system is to enable a user to generate effective and attractive illustrations of dynamic natural scene recordings from stereo as input.

2 ALGORITHM

Images of a natural, real scene usually contain texture, noise, or other features such as shadowing that set the accurate extraction of object boundaries to a complex task. A simple approach of using only the output of an edge detector for most real scene's images induces the problem of distinguishing object boundaries from the other features edges. To overcome this problem, one of the most valuable sources of information about

WSCG 2006 conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

Figure 1: Example 1 from the "Alan Ford" comic book series. © Copyright 2005 by Max Bunker Press. Used by permission.

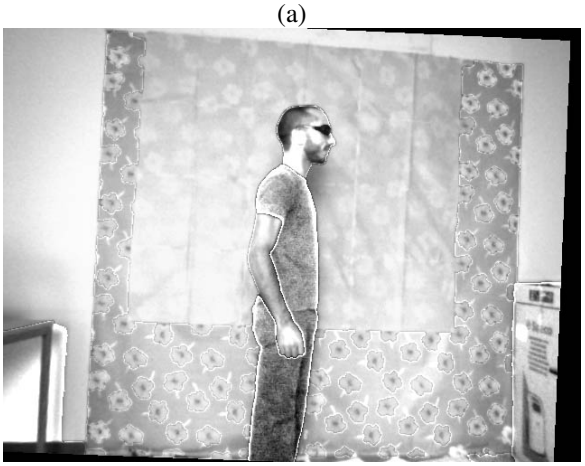
the objects present in the scene is the depth information. To obtain this information, from the images of the real scene, we use the depth-from-stereo approach. The output of the stereo matching process delivers a disparity map, which is directly related to the depth. In most cases, however, the object contours cannot be perfectly recovered from the disparity map alone, due to matching errors along the depth discontinuities. To suppress errors produced in the matching process we suggest the Edge Combination algorithm [MSG05], which uses disparity edges to identify corresponding edges in the original edge image to obtain more accurate information of the object boundaries. The result of this process, dominant edges that highlight the principal structure of the scene, is further used to stylize the image in a drawn-like form and to depict the motion through motion lines and multiple contours. Generally, our approach consists of two major steps: image representation in a drawn-like form and comics-like depiction of motion. These steps are further explained in the following sections.

2.1 Drawn-like image representation

As we previously mentioned, comic books are usually created by skilled and creative artists who use a language of visual symbols for drawing, unique for this type of art. A very strong distinctive feature of comics is a tight connection between a story and a style of drawing: this results in funny stories having characters drawn like caricatures or "saving the world" stories with superheroes drawn in a way to show the power. Our system does not intend to replace an artist, although the result can be used as final artwork, but to assist an artist in technical aspects and support him in routine tasks. It can also be very useful in a learning process because the final output preserves the position, proportion and shape of objects. Since we are dealing with recordings of a natural scene, the drawn representations keep the aspect of real.

To present an image in a drawn-like form, we make use of our previous work [MG05]. The computed stroke density varies according to the tone of the intensity image region, from dense strokes in dark to no strokes in light areas. To better convey the shape of objects in natural scene images, which usually suffer from a bad contrast, we first perform a local image contrast enhancement, which is driven by the color segmentation result of the original image. This produces an image with newly revealed details, as presented in Fig. 2. We then draw each stroke in a direction determined by the stereo derived disparity layers with the stroke density and distribution derived from a Poisson disc distribution. The disparity maps that we use in this work are results of the graph-cut based stereo-matching algorithm [BG05], which represents disparity by a set of planar layers. Such a representation of enhanced images using disparity to orient strokes results in a good distinction and shape depiction of objects in the scene.

(a)



(b)

Figure 2: (a) Original image, (b) enhanced original image, prepared for further stylization.

2.2 Tracking the motion

For tracking the motion of objects in the scene we employed the KLT tracker [ST94]. To depict motion in comics-like style, we need only information of distinct discontinuities in motion, which usually go along with depth discontinuities. By following this idea, among the point set selected automatically by the KLT tracker, we track only those points found to be close to edges in the Edge Combination image through all the frames of the user selected sequence.

2.3 Depicting motion

In comics, motion is usually represented using either motion lines, multiple contours or their combination, which results in a variety of styles. An example of motion depiction in a real comic through motion lines is given in Fig. 1, and using both motion lines and multiple contours is shown in Fig. 3.

Motion lines: also known as action lines or speed lines, is the most common style of representing motion in comic books. Depicting motion in comics has a meaningful purpose in story telling and strongly depends on an artist's imagination. Usually, motion for each object is presented through one path in several motion lines (see Fig. 1 and Fig. 3). In a natural scene, one or more objects can be sources of motion. Representing just the motion of the object with, for instance, the longest path would be a great limitation. As a remedy, we give control to the user to select a desired object and type of motion to depict by selecting one of the object's tracked points. Fig. 4 (b) shows a tracked point and its path selected from the tracking points in Fig. 4 (a). To preserve the usual simple motion illustration in comics, we fit primitives (i.e. circle, ellipse, straight line) to the path of the tracked point to get the final motion path and to have a possibility to extend its length. An example of fitting a primitive, in this case a circle, is shown in Fig. 4 (c). By further selecting points on the contour in the Edge Combination image, Fig. 4 (d), close to the tracked point, the user passes the same shape of the motion path to these points, and in this way more motion lines can be created from a tracked path. For a more natural, hand-drawn look, the number of points in the final motion path, for each motion line, is reduced by removing a small random number of points at the beginning of the path.

Multiple contours: is another, very important, style of motion representation which uses a complete or a part of a moving object's contour. To imitate this style we use Edge Combination images extracted from the frames in a regular interval and draw them on the final result. Regularity in frame selection gives a good impression of the object's motion speed. The variety of styles can be even larger by changing the transparency of contours through frames.

Figure 3: Example 2 from the "Alan Ford" comic book series. © Copyright 2005 by Max Bunker Press. Used by permission.

3 EXPERIMENTAL RESULTS

Example results of motion depiction using multiple contours, motion lines and their combination are given in Figs. 5 - 9. The initial stereo image pair of the image sequence, in epipolar geometry, is presented in Fig. 5 (a) and (b). Their resulting disparity image computed by employing the stereo matching algorithm presented in [BG05] is given in Fig. 5 (c), along with the labeled disparity layers shown in Fig. 5 (e). The color segmentation result, see Fig. 5 (f), is used in order to locally enhance the contrast to better convey the shape of the natural scene before further processing. Fig. 5 (h) illustrates the output of our algorithm in the style of motion depiction using motion lines together with the drawn-like representation of the natural scene from Fig. 5 (g). Another example of the same motion depiction style is given in Fig. 9 (c).

Two sets of images, presented in Fig. 6 and Fig. 7, demonstrate various styles of motion depiction through multiple contours and multiple contours joined with motion lines, respectively. The corresponding frames from the original sequence are shown in Fig. 8. The multiple contours styles that we illustrate in these figures are obtained by changing the transparency of contours through frames, using a complete or a part of a moving object's contour and their combination.

4 SUMMARY AND OUTLOOK

In this paper we have presented an algorithm to depict motion in comics-like form, with an artistic drawn-like representation of the scene, from a stereo image sequence. We have devised several motion depiction styles using motion lines and/or multiple contours. We have shown how silhouettes extracted by the Edge Combination algorithm can serve in motion depiction and also to portray the impression of the change of an object's speed.

In future steps, we intend to experiment with a spline representation of the motion path. This should give more flexibility and the possibility of depicting more complex motions, but it may also raise problems in motion path length extension. We also plan to experiment with other stylistic properties of comics and explore additional techniques of motion illustration in still images.

ACKNOWLEDGEMENTS

We wish to thank Michael Bleyer for providing the disparity maps. This research has been funded by the Austrian Science Fund (FWF) (project no. P15663) and by the Austrian Federal Ministry for Education, Science, and Culture, as well as the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

REFERENCES

[BE01] Gabriel J. Brostow and Irfan Essa. Image-based motion blur for stop motion animation. In *Proceedings of SIGGRAPH '01*, pages 561–566, August 2001.

[BG05] Michael Bleyer and Margrit Gelautz. Graph-based surface reconstruction from stereo pairs using image segmentation. In *Proceedings of SPIE*, volume 5665, pages 288–299, 2005.

[CRH03] John P. Collomosse, David Rowntree, and Peter M. Hall. Cartoon-style rendering of motion from video. In *Proceedings of Video, Vision and Graphics (VVG)*, pages 117–124, July 2003.

[Cut02] James E. Cutting. Representing motion in a static image: constraints and parallels in art, science, and popular culture. *Perception*, 31(10):1165–1193, 2002.

[Har99] Robert C. Harvey. *Children of the Yellow Kid: The Evolution of the American Comic Strip*. University of Washington Press, 1999.

[KE05] Byungmoon Kim and Irfan Essa. Video-based nonphotorealistic and expressive illustration of motion. In *Proceedings of Computer Graphics International (CGI '05)*, pages 32–35, June 2005.

[McC00] Scott McCloud. *Reinventing Comics: How Imagination and Technology Are Revolutionizing an Art Form*. Harper Paperbacks, 2000.

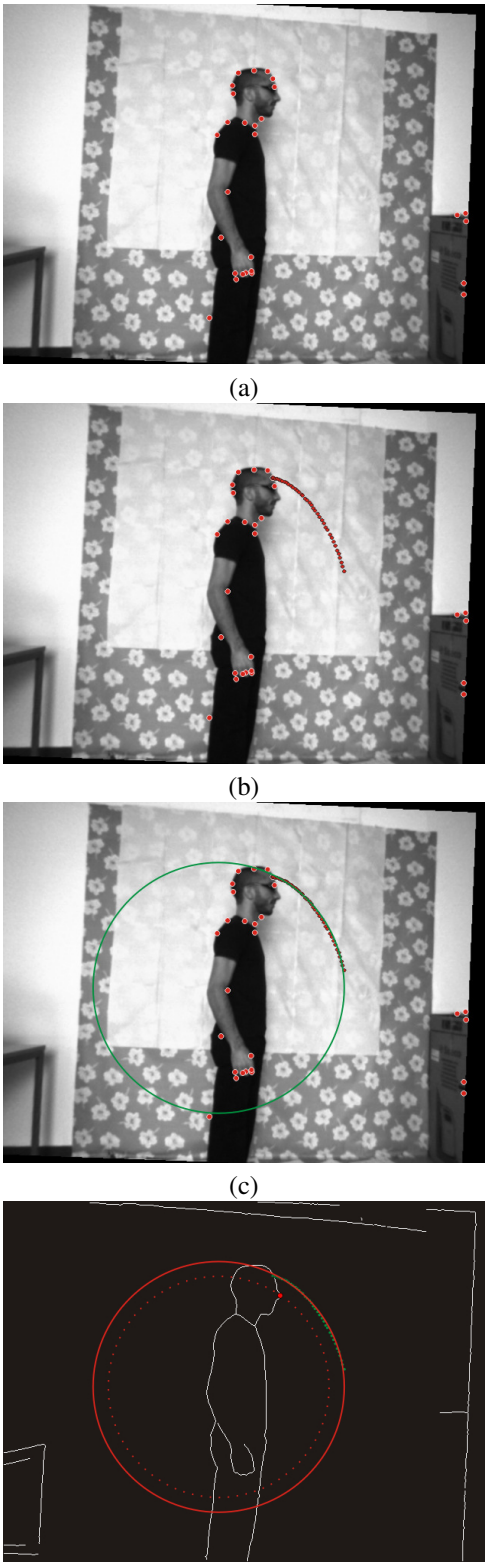
[MG05] Danijela Marković and Margrit Gelautz. Drawing the real. In *Proceedings of GRAPHITE'05*, pages 237–243, November 2005.

[MSG05] Danijela Marković, Efstathios Stavrakis, and Margrit Gelautz. Parameterized sketches from stereo images. In *Proceedings of SPIE*, pages 783–791, January 2005.

[MSS99] Maic Masuch, Stefan Schlechtweg, and Ronny Schulz. Speedlines: depicting motion in motionless pictures. In *Proceedings of SIGGRAPH '99: ACM SIGGRAPH 99 Conference abstracts and applications*, page 277, August 1999.

[Sab01] Roger Sabin. *Comics, Comix & Graphic Novels: A History Of Comic Art*. Phaidon Press, 2001.

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, June 1994.



(a) (b) (c) (d)

Figure 4: (a) Points to track, (b) motion path of the selected point, (c) fitted circle to the motion path, (d) selected new point on the contour in the Edge Combination image and its path.

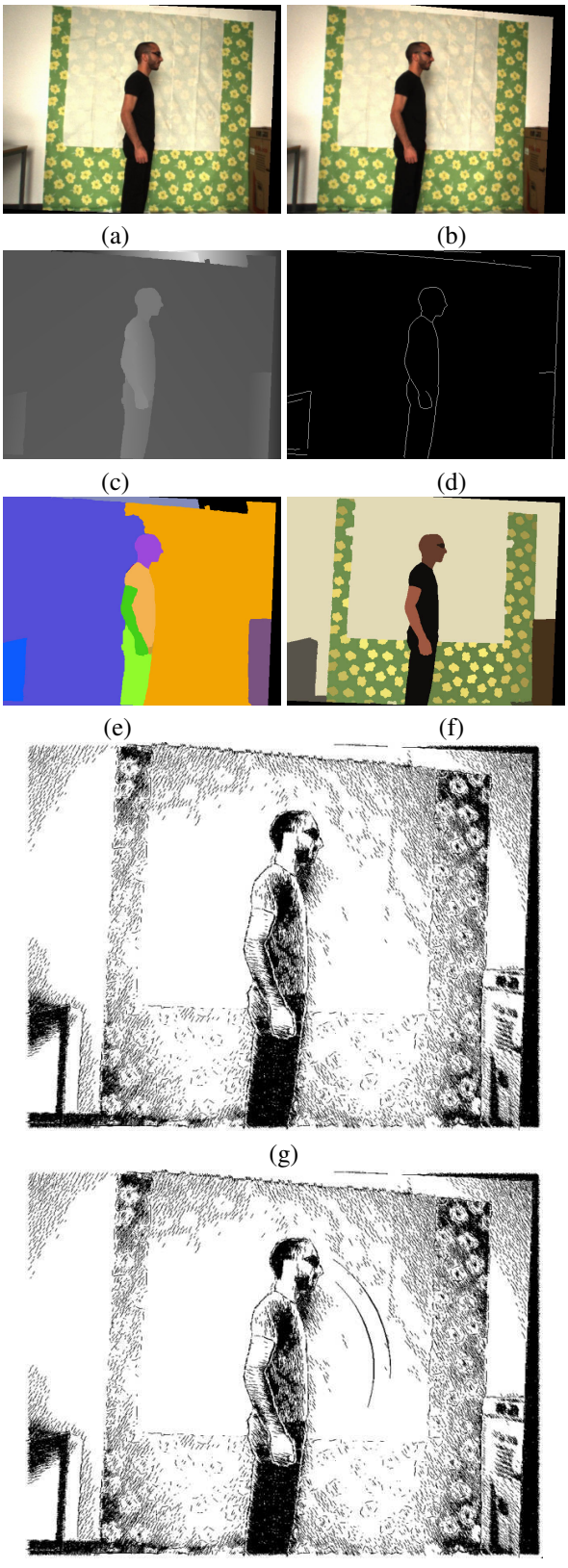


Figure 5: (a) Left camera image, (b) right camera image, (c) disparity image, (d) Edge Combination image, (e) layers image, (f) color segmented image, (g) drawn image, (h) drawn image with depicted motion.

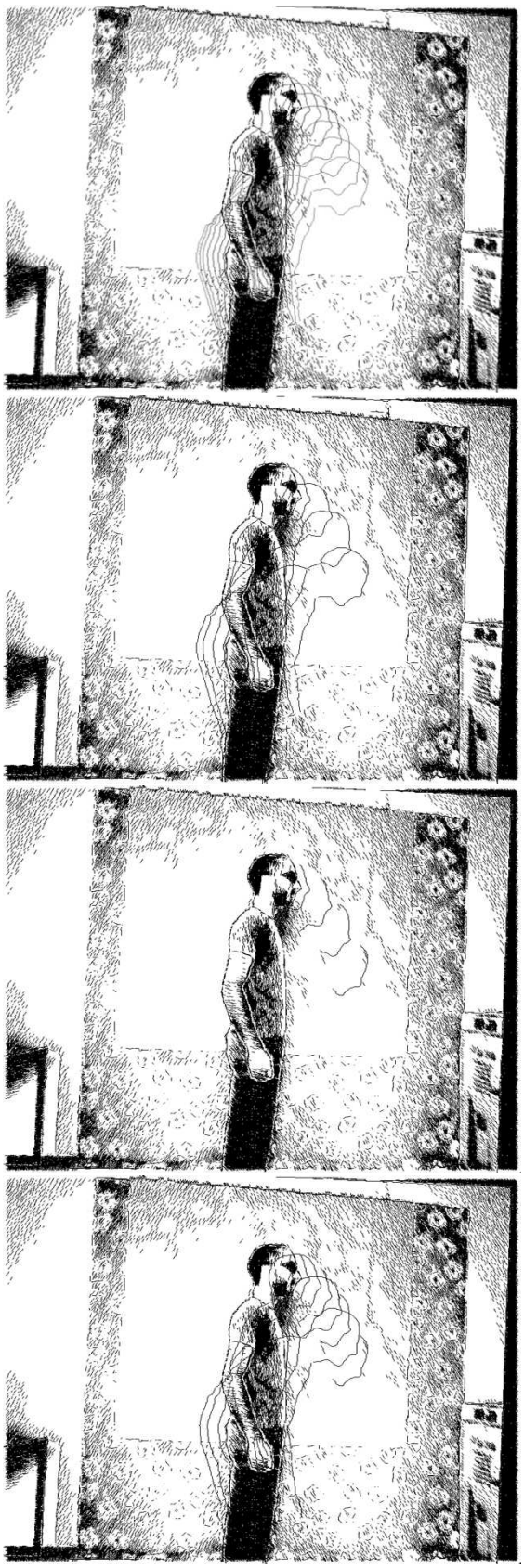


Figure 6: Results of motion depiction through multiple contours.

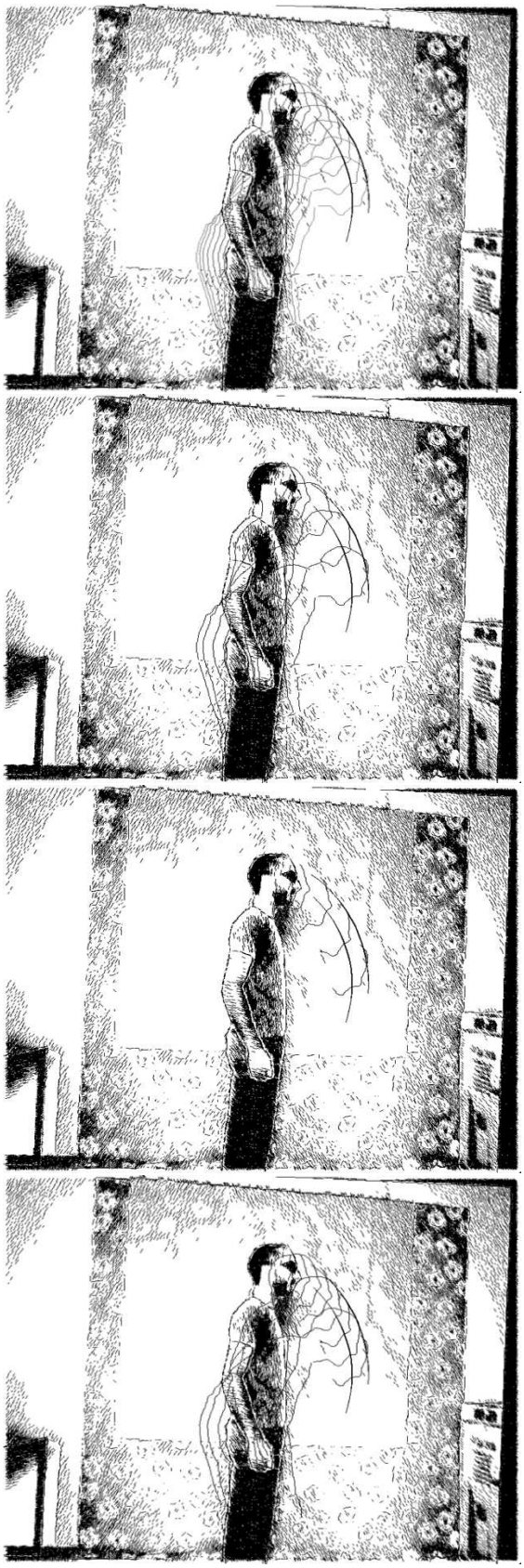


Figure 7: Results of motion depiction through multiple contours and motion lines.

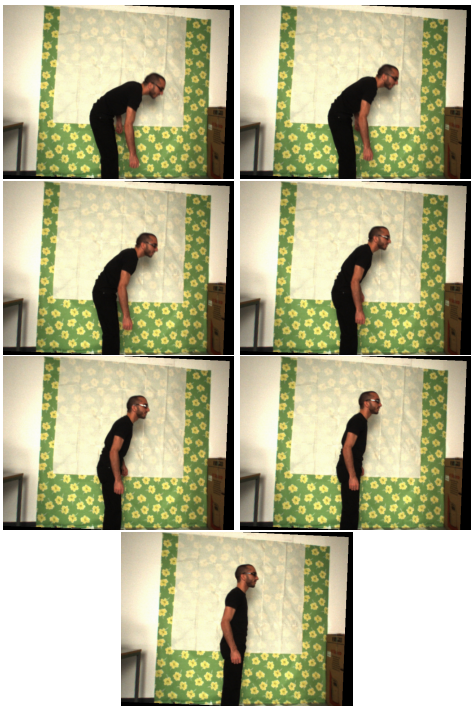


Figure 8: Intermediate frames (from top left to bottom) used for computation of the contours in Figs. 6 and 7.

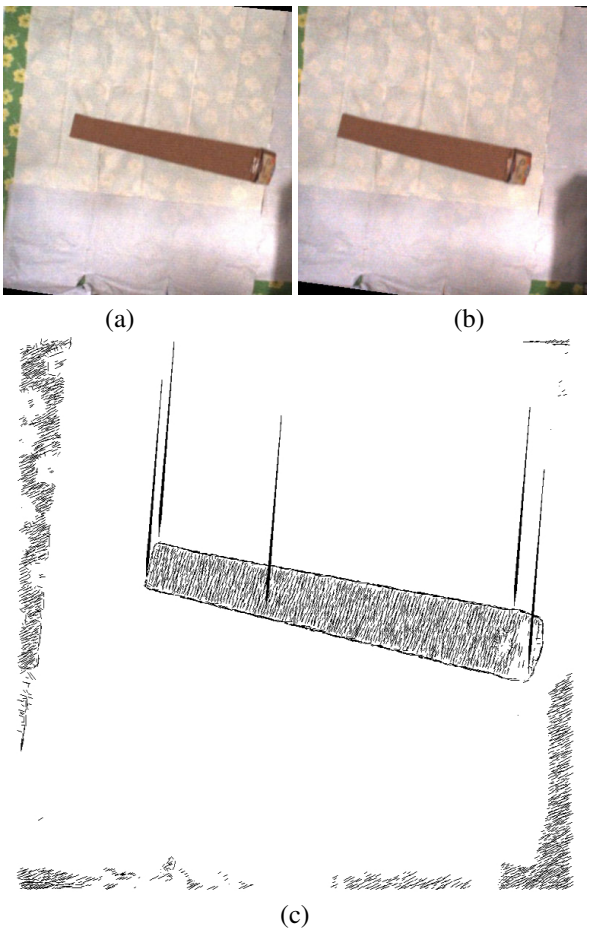


Figure 9: (a) Left camera image, (b) right camera image, (c) drawn image with depicted motion.

Video-Based Rendering Of Traffic Sequences

Cedric Vanaken, Tom Mertens and Philippe Bekaert

Hasselt University — Expertise Centre for Digital Media
and transnationale Universiteit Limburg — School of Information Technology
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{cedric.vanaken, tom.mertens, philippe.bekaert}@uhasselt.be

ABSTRACT

Video-based rendering is a viable alternative to traditional realistic image synthesis techniques. It avoids the burden of time-consuming modeling and expensive global illumination simulation. In this paper we propose a video-based synthesis and animation system for fixed viewpoint scenes that feature rigid objects. We examplify this by using traffic video sequences. In a first stage, we extract vehicles and their trajectories from the example footage using an intuitive semi-automatic segmentation technique. Subsequently, the resulting vehicle "sprites" are dimensionally reduced using PCA in order to effectively extrapolate trajectories from incomplete sequences. Background, occlusions and shadows cast by vehicles are extracted as well. We show that convincing new footage can be synthesized readily from a single input video. Any number and variety of cars can be inserted, and their trajectories can be edited to simulate such traffic scenarios as lane changes and traffic jams.

Keywords: Video-based Rendering, Animation, Traffic, Video Sprites.

1 INTRODUCTION

Video-based rendering methodologies have proved to be adept at synthesizing photo-realistic video sequences from sparse real world data, e.g. Schödl et al. [SSSE00]. In the same spirit, we present a novel technique to synthesize traffic scenes. Given an input traffic video, we are able to reproduce a traffic scene from the same viewpoint in which the configuration and trajectories of the vehicles have been altered by a user.

The main application of our technique is the validation and training of camera-based traffic analysis, e.g. for accident, queue and presence detection [Tra]. These systems cannot afford to have a high margin of error and thus require a wide variety of initial test sequences. Because these sequences are unique for each camera placement, they usually have to be acquired by shutting down highways and filming all desired scenarios *in situ*. This is an expensive and time-intensive task. As an alternative, one might synthesize video sequences directly using traditional modeling and global illumination techniques [DBB03]. However, this is an overwhelming task, both in terms of manual labor and computational requirements. Moreover, one would need to achieve a degree of realism that is hardly practical using current modeling and rendering tools (e.g. to reproduce natural landscapes, weathered surfaces, etc...). Also, it is unclear how imperfections of the camera system would be simulated. By using recorded footage from the targeted camera system, we achieve both goals immediately. Aside from validation of camera-based traffic detection systems, we believe that the techniques developed for this particular problem are general enough to be applied to video-based sprite animation [SE02].

Akin to Debevec et al.'s image based modeling approach [DTM96], one might construct approximate geometry for each vehicle of interest. However, this will turn out to be a labor-intensive task, since this process has to be carried out for each car separately. Alternatively one might come up with a parameterized model for vehicle geometry which can be fitted to the image data [BV99]. It is unclear whether such a model can be general enough to deal with the wide variety of shapes found in real life.

The main challenge is to extract vehicle sprites from the input footage and resynthesize them in a meaningful and controllable fashion [SSSE00, SE02]. These sprites are 2D images that represent an object of interest that can afterwards be integrated at different locations in the input scene, or inserted into a novel scene. We assume vehicle sprites travel along a straight path, while Video Sprites [SE02] aims at reconstructing arbitrary motions. This prior information is exploited in our segmentation and easily allows for parameterizing a vehicle's trajectory and appearance. In contrast, Video Sprites are a non-parametric representation based on searching and copying the most suitable frame from the input data. The parameterized representation facilitates easy extrapolation of incomplete trajectories (e.g. when the vehicle is not completely visible), and is friendly toward storage.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30 - February 3, 2006
Plzen, Czech Republic.

In addition, we need to extract the background, which we assume to be static, and deal with possible occlusions along a vehicle's trajectory (e.g. caused by a bridge or lamp post). Schödl et al. [SSSE00, SE02] record sprite images and accompanying shadows using chroma keying. In our setting, this information cannot be extracted under such controlled conditions.

Jojic et al.'s video sprite model [JF01] copes with inter-sprite occlusions efficiently. For our purposes this is less of an issue, but static obstructions like lamp posts have to be dealt with.

The outline of our system is presented in Figure 1.

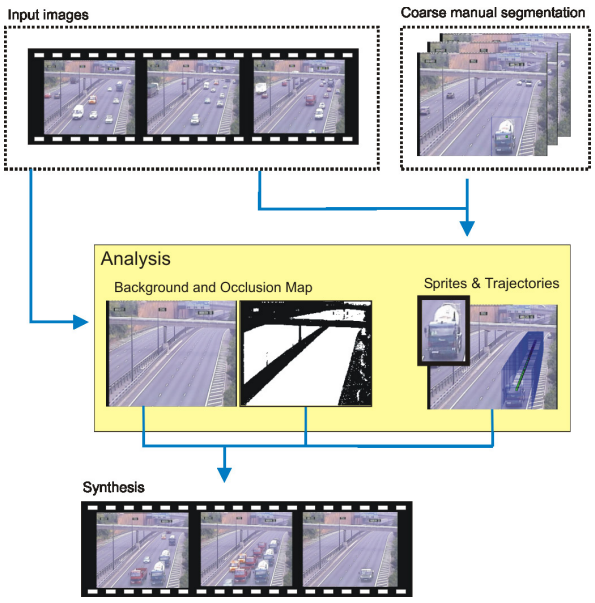


Figure 1: Outline of our traffic animation system. Starting from an input video and some user assistance, the analysis phase outputs a background, an "occlusion map" and extracted vehicles with associated trajectories. This data is used as input for the synthesis phase, where the vehicles are drawn onto the background to obtain a new animated video.

The rest of the paper is organized as follows. We start by reviewing related work in the areas of traffic detection techniques and video based animation systems. Section 2 gives an outline of our system. Section 3 presents our results and discusses practical issues. Finally, in section 4 we will present our conclusions and suggestions for future work.

1.1 Related Work

Video-based representation, rendering and animation techniques have received increasing interest in the graphics and vision community in the past few years. A recurring technique is rearranging frames in an input video to create novel footage, either globally on a per-frame basis [BCS97, SSSE00, AZP+05], or locally on a per-sprite basis [SSSE00, SE00, SE02]. This approach allows for efficient animation of e.g. natural phenomena or animals. Our approach differs by not reordering frames but rather building a simple parametric motion and appearance model for vehicle sprites, similar in spirit to Fitzgibbon [Fit01]. Consequently, less input is required and this representation even facilitates extrapolation.

Layered video models [WA94, JF01] automatically extract layered sprites and moving parts from input footage. In our particular problem the layers are fixed: background, vehicle sprites and occluders (such as a lamp post). Background and occluders are automatically recovered, while a user assisted process is employed to extract the vehicles.

The flow-based video synthesis technique [BSHK04] analyzes the motion of textured particles in the input video along user-specified flow lines, and synthesizes video of arbitrary length by enforcing temporal continuity along a second set of user-specified flow lines. The main difference between this approach and ours is that we not only redraw input pixels, but also capture and reuse the entire appearance of the objects in the input video.

Auto-regressive stochastic processes [CV05, CV06] model traffic flow from video. This method does not require segmentation or tracking, but lacks the per-vehicle control that our approach offers.

Segmentation and tracking of vehicles is a central problem in traffic analysis [ZCSP03, CGPP00, CBMM98, KM03], which has to be fully automated. We opted for a simple and robust semi-automatic system, though in a more general settings, these techniques could be used as well.

2 SYSTEM OVERVIEW

Our system consists of an analysis stage and a synthesis stage, which will be detailed in the following sections.

2.1 The Analysis Stage

The analysis stage extracts a background image, an occlusion map and vehicle sprites together with their trajectories.

Background We obtain an appropriate background as the per-pixel median intensity along the time dimension [GASK95]. This simple technique works very well on our video sequences, which have virtually static backgrounds. This rendered the implementation and testing of other, more complicated, techniques (e.g. [SG99]) unnecessary for us. An example of a calculated background image is shown on the left side of Figure 3.

Occlusion Map The occlusion map indicates which pixels remain unchanged during the entire length of the input video. Those pixels that fluctuate significantly w.r.t. some threshold T can be considered "possibly foreground", here the background should always be

Figure 2: Windows drawn around a particular vehicle by the user, with on the right a calculated trajectory for the vehicle.

drawn behind the sprites. A static pixel will either be an occluder (for example the bridge in Figures 2 and 3) or just background, and should be drawn on top of the sprite layer. We empirically estimated T to an intensity value of 0.012 in our experiments. If necessary, the occlusion map can be edited easily with any image editing tool to clean any impurities. We have made use of this feature to correct a very small number of pixels that slipped through the threshold. See Figure 3 for an example occlusion map.

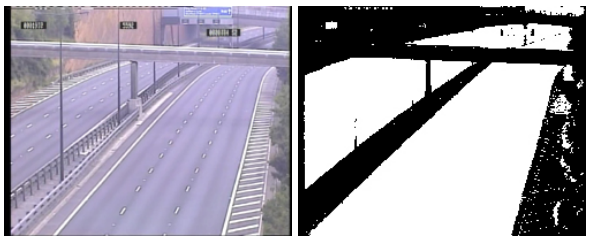


Figure 3: Left: extracted background image from an input video. Right: occlusion map. Static pixels (occluder or background) are black and fluctuating pixels (possibly foreground) are white.

Segmentation Extracting a vehicle from the input video is a semi-automatic process, which starts with a small amount of user interaction. The user takes three frames in which a particular vehicle can be seen at different positions: one frame where the vehicle has initiated its trajectory, one where it is approximately half way, and one near the end. The user indicates a window around the vehicle in each of these three frames. The content of this window captures all relevant information about the vehicle: its appearance and surrounding illumination effects (i.e. shadows). This window does not need to be drawn very precisely, it is only necessary that the vehicle and illumination effects are included and that no other vehicles or parts of other vehicles appear inside the window. The position of the window also defines where the vehicle is located at a known instance in time (fig 2).

Vehicle Sprite Appearance Model In this section we detail the vehicle appearance model, which is based on the user masks and pixel intensities in the window.

Assuming that the vehicle moves at a near constant speed and travels along a fairly straight line, we can interpolate the windows for the intermediate frames in time (t) by fitting the following simple parametric function to each of their corner positions x:

$$x(t) = \frac{(a \times t) + b}{t + c}$$

Thus, we can model the vehicle's trajectory with a simple function that takes into account perspective projection.

In the next step, the user draws a mask for the vehicle in the three initially indicated windows (see Figure 4).

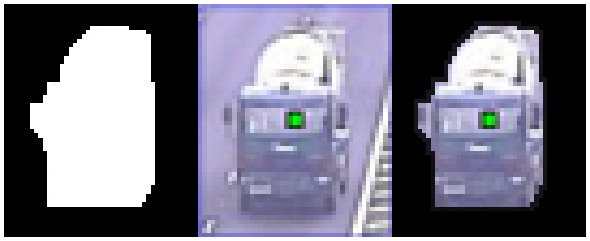


Figure 4: Left: user drawn mask. Middle: vehicle for which the mask is drawn. Right: result of the mask operation.

In a more general setting, this manual intervention may be replaced with an automatic technique.

In order to obtain the masks for intermediate frames, we convert the masks to polygonal shapes and interpolate them. However, the number of vertices for different masks is not guaranteed to be the same. Inspired by the morphing algorithm of Kent et al. [KCP92], we solve this problem as follows. Let A, B and C be the 3 user defined mask polygons. Polygon A is placed on B and we project and add each of A's vertices to B. This step is repeated from A to C, B to A, etc...We are able to reconstruct the mask for each frame simply by rasterizing the corresponding interpolated polygon. In addition, we soften the edges of the binary masks using convolution to avoid possible seams between vehicle and background. The results of this simple matting technique, are qualitatively the same for our input videos as results that can be achieved by using more advanced matting techniques [CCSS01, SJTS04].

The vehicle's appearance inside the mask mainly evolves due to a small relative rotation w.r.t. the camera and environmental illumination (e.g. from street

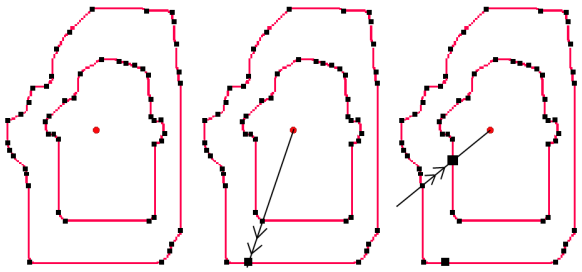


Figure 5: Left: source polygon placed on target polygon. Middle: extra interpolation step. Right: inverse direction extra interpolation step.

lights). In addition to pixels belonging to the vehicle itself, shadow information is extracted by dividing the window content by the background. This yields a background-invariant multiplicative "shadow map", as seen in Figure 6. Naively storing full windows is dependent on the background, and might corrupt the appearance when altering the location of a vehicle. We therefore also blur out all detail surrounding the mask in the vehicle sprite (not done for the shadow map).

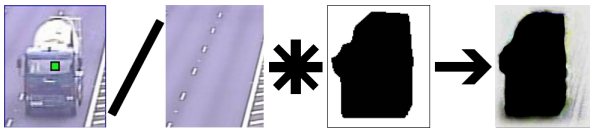


Figure 6: Shadow map computation (right image is contrast enhanced).

Given the full appearance of a vehicle at each frame, we reduce it using PCA. Before we do this, we need to scale our masks, sprites and shadow maps to a common size, for which we take the maximum window size of the indicated windows. Then, the polygon vertices relative to the midpoint, the sprites' pixel intensities and the shadow map intensities are each concatenated into a long vector with dimensionality V. Usually V is quite large (e.g. 10^5) while the number of frames N is small (e.g. 100), yielding a $V \times N$ matrix D that features an impractically large covariance matrix. We follow Matusik's variant on PCA [Mat03] to circumvent this problem. More precisely, we perform an eigenvalue decomposition of the $N \times N$ dimensional covariance matrix of the zero mean D_0. The obtained eigenvectors (*eivec*) are sorted by ascending eigenvalues (*eival*), while the vectors with very low eigenvalues are thresholded. Finally, our PCA representation of the appearance of the vehicles consists of:

- transformation matrix: $T = D_0 \times eivec \times (eival^{-\frac{1}{2}})$
- PCA coefficients matrix: $X = T \times D_0$
- mean image of the frames: μ

The eigenvectors corresponding with the highest eigenvalues contain the coarse details, while subsequent values express the finer details. Reconstruction quality can be traded off against the level of compression by discarding eigenvectors that have a small eigenvalue associated with it. We found that the 20 (out of 10^5 in total) largest eigenvalues and accompanying eigenvectors are sufficient to reconstruct a sprite's appearance.

Usually, we cannot capture a vehicle's full trajectory on the screen, because the sprite is cropped at the edge of the screen near the beginning and end, or possibly occluded. The inclusion of these sprites in the input data will result in a somewhat distorted PCA result. Therefore, we don't take these frames into account, but solve this problem by extrapolating the PCA coefficients that parameterize the vehicle's appearance, using autoregression [Fit01, NS04]. These "fitted" PCA coefficients will be used in the synthesis step to complete the trajectory of the vehicles at points in time where the vehicle was not segmented.

A comparison of our extrapolation technique with ground truth is shown in Figure 7. From this we can conclude that the shape of the vehicle is approximated fairly well, while interlacing artefacts are partly smoothed out by the PCA algorithm.

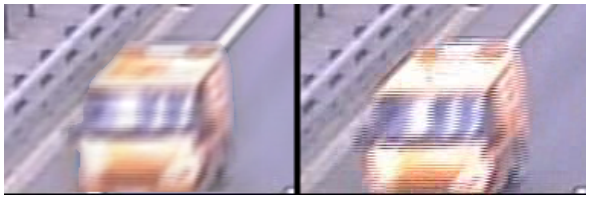


Figure 7: Left: extrapolated vehicle at the end of its trajectory. Right: Ground truth.

2.2 The Synthesis Stage

The synthesis step is relatively straightforward. We initialize the frame buffer with the background image. For each vehicle and a given frame we need to perform the following steps, of which only number 1 needs a more in-depth explanation.

1. Compute the vehicle's appearance.
2. Scale the vehicle to its original window size
3. Paste the vehicle onto the frame buffer
4. Multiply the frame buffer with the shadow map.

The sprites are rendered in back to front order to correctly resolve inter-vehicle occlusion.

Step 1 consists of rebuilding the appearance of the vehicle out of the PCA representation. We start off with the mean image μ, and iteratively add more detail. We do this by reshaping the next-in-line eigenvector from the transformation matrix T to an eigenimage, multiplying it with its associated PCA coefficient and adding

the result to the mean image. Figure 8 shows some examples of intermediate results of this iterative process. We have empirically fixed the number of used PCA levels to 20.

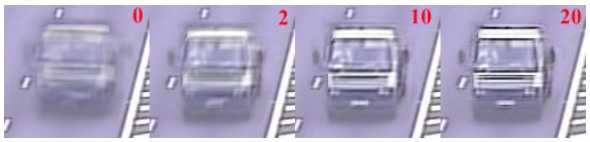


Figure 8: Iterative reconstruction from PCA data: the first image shows the mean appearance of the vehicle without extra detail added to it. The next images show the mean image with respectively 2, 10 and 20 PCA levels added. It is clearly visible that the first PCA levels contain the most important information, while the lower levels add only little detail.

3 RESULTS AND DISCUSSION

We extracted five different vehicles from a short input video and out of this data synthesized four videos that display different animated traffic situations:

- normal traffic
- a traffic jam
- a vehicle that suddenly stops while the rest of the traffic continues on the other lanes
- a vehicle that drives backwards while the rest of the traffic drives normally on the other lanes

The extracted vehicles that were used for the rendering of these new videos are shown in Figure 9.

Figure 9: Extracted vehicles, scaled to the same height for presentation purposes.

The amount of user-interaction involved in the creation of these results is fairly low. In the analysis stage, three rectangles have to be drawn around each vehicle, which typically takes a few seconds. The most time-consuming step is drawing a mask for each rectangle. This task may require a couple of minutes per mask for an unexperienced user. Note that these steps need to be performed only once per segmented vehicle.

Due to memory restrictions in MATLAB we were unable to simultaneously load more than five vehicles into memory. As a quick workaround for this problem, we resorted to using the same vehicles more than once in our synthesis progress. Another option, in a commercial context, would be to introduce a database where the vehicles could be stored and queried in their compact representation.

Our synthesized videos suffer from such artefacts as interlacing and motion blur. These artefacts are already present in the input video (as can be seen in Figure 9), so it is only logical that they also occur in our output videos. As our algorithm was developed for detection systems, this actually becomes a major advantage. Synthesized videos that look too polished provide an unrealistic training test for these systems that does not correspond with the real world conditions.

In the accompanying videoclips (available on the CD-ROM), we can clearly see that thanks to our occlusion map, all occlusions are handled correctly. The videos show vehicles departing from underneath a bridge, where they are correctly hidden from view thanks to the occlusion map. In frames where the vehicles are only partially visible, we apply extrapolation of the PCA components to obtain an approximation of the entire vehicle. The shape and appearance of remote vehicles are accurately estimated using our autoregression algorithm as their orientation barely changes. Closer to the camera however, the orientation of the vehicles can vary rapidly w.r.t. the camera. Autoregression has a hard time estimating changes before they occur because its prediction is based on previous frames. This may cause a slightly thicker edge around the vehicle. This problem however only occurs on a few synthesized vehicles and is visible in just a small area in a few frames, rendering its impact on traffic detection systems negligible.

Figure 10: An edge that might appear around a vehicle when its orientation suddenly changes.

The trajectories of the vehicles in the synthesized videos are restricted to their original trajectories in the input video. This is because the camera has a different perspective view on each lane. If a vehicle is synthesized on a different lane than the one it originally occupied, it will be perceived to be sliding on the road surface instead of driving down the road. However, slight deviations are allowed.

The effectiveness of our shadow maps in synthesizing the illumination effects around the vehicles is illustrated in Figure 11.

Figure 11: Left: synthesized vehicle using a shadow map. Right: synthesized vehicle without using a shadow map.

4 CONCLUSION AND FUTURE WORK

In this paper, we presented a novel video based method for rendering and animating rigid objects in fixed viewpoint video scenes. Our method was examplified by using traffic video sequences. A parameterized sprite appearance model is central in our approach. It describes how the sprite evolves in shape and pixel intensity, and also allows for interpolation, extrapolation and compact storage. Using this information, we can synthesize new videos that feature these sprites, in such a way that the videos exhibit animated traffic situations.

We would like to explore approximate geometric representations to more rigorously represent the relative rotation of the vehicles. Furthermore we believe that variable weather conditions and intricate illumination effects like vehicle headlights can be a valuable addition to our framework.

New trajectories for vehicles could be synthesized if control over the input of the videos is available. A vehicle filmed driving on several different lanes may then be interpolated horizontally afterwards.

Further research is also needed for solving the thick edges that sometimes appear around vehicles near the end of their trajectories. Different prediction and estimation techniques may need to be investigated for this.

ACKNOWLEDGEMENTS

The authors acknowledge financial support on a structural basis from the ERDF (European Regional Development Fund), the Flemish Government and the Flemish Interdisciplinary institute for BroadBand Technology (IBBT), and from research grants by the EU (IST-2-511316-IP "Racine-IP") and the IWT (innovation feasibility study 040658 with Traficon NV). We would like to thank Traficon for making available their traffic video sequences.

Furthermore we would like to thank our colleagues for their help and inspiration, especially Mark Gerrits, Tom Haber and Erik Hubo.

REFERENCES

[AZP+05] A. Agarwala, K.C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski. Panoramic video textures. *ACM Trans. Graph.*, 24(3):821–827, 2005.

[BCS97] C. Bregler, M. Covell, and M. Slaney. Video rewrite: driving visual speech with audio. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 353–360, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[BSHK04] K.S. Bhat, S.M. Seitz, J.K. Hodgins, and P.K. Khosla. Flow-based video synthesis and editing. *ACM Trans. Graph.*, 23(3):360–363, 2004.

[BV99] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH'99 Conference Proceedings*, 1999.

[CBMM98] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research: Part C*, 6(4):271–288, 1998.

[CCSS01] Y. Chuang, B. Curless, D.H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271. IEEE Computer Society, December 2001.

[CGPP00] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Statistic and knowledge-based moving object detection in traffic scenes. *Proc. of ITSC-2000 - The 3rd Annual IEEE Conference on Intelligent Transportation Systems*, pages 27–32, Oct. 1-5, 2000.

[CV05] A.B. Chan and N. Vasconcelos. Classification and retrieval of traffic video using auto-regressive stochastic processes. *IEEE Intelligent Vehicles Symposium*, 2005.

[CV06] A. Chan and N. Vasconcelos. Layered dynamic textures. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

[DBB03] P. Dutré, P. Bekaert, and K. Bala. *Advanced Global Illumination*. AK Peters, 2003.

[DTM96] P. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH Proceedings*, 1996.

[Fit01] A. Fitzgibbon. Stochastic rigidity: Image registration for Nowhere-Static scenes. *Proceedings of International Conference on Computer Vision*, pages 662–669, 2001.

[GASK95] B. Gloyer, H. Aghajan, K.-Y. Siu, and T. Kailath. Video-based freeway monitoring system using recursive vehicle tracking. In *Proceedings of SPIE*, February 1995.

[JF01] N. Jojic and B. Frey. Learning flexible sprites in video layers. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[KCP92] J.R. Kent, W.E. Carlson, and R.E. Parent. Shape transformation for polyhedral objects. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 47–54, New York, NY, USA, 1992. ACM Press.

[KM03] Z. Kim and J. Malik. Fast vehicle detection with probabilistic feature grouping and its application to vehicle tracking. In *In Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2003.

[Mat03] W. Matusik. *A Data-Driven Reflectance Model*. Massachusetts Institute of Technology, 2003.

[NS04] A. Neumaier and T. Schneider. Arfit, a matlab package for the estimation of parameters and eigenmodes of multivariate autoregressive models. *ACM Trans. Math. Softw.*, 27:27–57, 2004.

[SE00] A. Schödl and I. Essa. Machine learning for video-based rendering. In *Proceedings of NIPS*, 2000.

[SE02] A. Schödl and I. Essa. Controlled animation of video sprites. *Proceedings First ACM Symposium on Computer Animation*, July 2002.

[SG99] C. Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition*, pages 246–252, 1999.

[SJTS04] J. Sun, J.J., C. Tang, and H. Shum. Poisson matting. *ACM Trans. Graph.*, 23(3):315–321, 2004.

[SSSE00] A. Schödl, R. Szeliski, D.H. Salesin, and I. Essa. Video textures. *Proceedings of SIGGRAPH*, pages 489–498, 2000.

[Tra] Traficon, website: `http://www.traficon.com/`.

[WA94] J.Y.A. Wang and E.H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing Special Issue: Image Sequence Compression*, 3(5):625–638, September 1994.

[ZCSP03] C. Zhang, S-C. Chen, M-L. Shyu, and S. Peeta. Adaptive background learning for vehicle detection and spatio-temporal tracking. *IEEE Pacific-Rim Conf. on Multimedia (PCM'03)*, 2003.

Bandwidth-efficient Hardware-Based Volume Rendering for Large Unstructured Meshes

Thierry Carrard
CEA/DIF
DSSI
BP 12
France, 91680 Bruyères-le-Châtel
thierry.carrard@cea.fr

Manuel Juliachs
Laboratoire PRiSM
Université de Versailles-Saint-Quentin
45 avenue des Etats-Unis
France, 78035 Versailles
mju@prism.uvsq.fr

ABSTRACT

Recent advances in graphics processor architecture and capabilities have made the development of fast and efficient unstructured volume rendering methods possible. These techniques can be classified into two roughly delimited categories: cell projection based methods and GPU raycasting algorithms. However, both approaches are subject to limitations, respectively due to the main memory-to-GPU bandwidth for the former and due to the GPU per-fragment computation speed and memory size for the latter. These potential bottlenecks can be particularly limiting for large-size datasets, such as the ones produced by large-scale numerical simulation. In this work, we describe an enhancement to the cell-projection rendering method, allowing us to specify each tetrahedron with only 4 vertices and their associated data. By using a point sprite primitive, instead of a set of 4 triangles, we significantly reduce the amount of data transferred from the main memory through the graphics port for each frame rendered. We evaluate the impact of the different rendering stages of our method on the overall frame rate.

Keywords

unstructured meshes, volume visualization, GPU-based rendering.

1. INTRODUCTION

Numerical simulations of unsteady physical phenomena yield datasets comporting a very large number of elements. We are interested in such datasets, usually sharing the following characteristics:

- a very large number of elements (typically ranging from 10^6 to 10^8 elements),
- unstructured meshes, with tetrahedra, hexahedra or other types of cells,
- a large number of different time steps,
- a high dynamic range, both in time and space.

Scientific visualization is a way to gain insight into the simulated phenomena. However, efficiently visualizing such datasets requires high performance techniques and methods. Volume rendering of unstructured datasets is an example of such an advanced visualization technique. Recently, performance and functionalities of commodity graphics processors have reached a point enabling the implementation of complex volume rendering algorithms, dramatically accelerating their performance with respect to previous software implementations. Current graphics processors (or GPUs) are able to process several hundred millions of vertices per second and several billions fragments per second [Nvi04a], allowing relatively complex user-defined programs to be applied to each processed vertex and fragment.

Taking advantage of this dramatic performance increase, several GPU-based unstructured volume rendering methods have been developed in the past years, allowing to render approximatively between 500k and 1 million tetrahedra per second, processing only relatively modest-sized datasets. However, given the current size of the datasets routinely produced by numerical simulations, GPU-based volume rendering methods should be able to render at least 10 million elements per second to meet the visualization needs of computational scientists.

Conference proceedings ISBN 80-86943-03-8
WSCG'06, January 30-February 3, 2006
Plzen, Czech Republic.

The projected tetrahedra (PT) technique [Shi90] is a well-known hardware-based unstructured cell-projection volume rendering method. It requires cells to be rendered according to a correct visibility order (either front-to-back or back-to-front), which is typically determined on the CPU, prior to rendering, using a sorting algorithm. Volume cells are decomposed on the CPU into a set of transparent triangles, which in turn are sent to the graphics card and blended into the framebuffer.

Recently, several enhancements of this method have been proposed, taking advantage of the programmability features afforded by modern GPUs. These methods generally allowed to bypass the CPU tetrahedron decomposition phase, resulting in an additional performance gain. However, most of these methods require that each tetrahedron be transmitted as a set of four triangles, which can result in a very high bandwidth consumption between the main memory and the graphics card. The ability to specify volume primitives with existing graphics APIs, as suggested by King *et al.* [Kin00a] would allow to go beyond this restriction.

In this work, we present a GPU-based volume rendering method, building upon previously developed approaches. First, we describe related work in the field of GPU-based unstructured volume. In the following section, we describe in detail our point sprite-based GPU volume rendering method, which optimizes the bandwidth usage by reducing the amount of data sent for each tetrahedron. Then, we present an adjacency search method, partly executed on the GPU. After that, we present some experimental results and a comparison with previously published approaches. Finally, we discuss some limitations of our implementation.

2. RELATED WORK

The scope of this section is hardware-accelerated volume rendering. We classify the different approaches into object-space or image-space rendering, the OpenGL rendering pipeline being an example of such the former whereas raycasting is typical of the latter.

In order to compute a correct image, graphics rendering usually requires to depth-sort primitives according to their viewpoint distance. Surface rendering generally only requires to find to nearest primitive (except if transparent). However, volume rendering requires to find all the primitives intersected by a given view ray (except if opaque objects allowing early termination are encountered). Depending on the optical model used [Max95a], this requires a visibility ordering of the primitive set intersecting any given ray. This visibility ordering can be done either in objet-space or in image-space.

In unstructured volume rendering, object-space methods first sort primitives according to the distance to the viewpoint, then render each primitive individually in the visibility order, accumulating their contributions into the frame buffer. Image-space methods process each view ray sequentially. For a given ray, all the intersecting primitives are determined, ray segments are computed and sorted into the correct visibility order and finally accumulated to determine the final pixel color.

Shirley *et al.* [Shi90a] developed the first hardware-accelerated unstructured volume rendering method. Their projected tetrahedra (or PT) method used the alpha-blending capabilities of the then existing graphics boards to accelerate the rendering of tetrahedral cells. In this method, each tetrahedron is projected on the view plane and decomposed into a set of (up to four) non-overlapping triangles, according to a projection class depending on the point of view, with a thick vertex at the longest ray segment-tetrahedron intersection. The triangles are then transmitted to the graphics accelerator and rendered into the frame buffer using alpha-blending to implement transparency. The thick vertex color and opacity are determined by a transfer function and hardware linear interpolation is used to compute color and opacity at each rasterized fragment. However, opacity usually does not vary linearly across a tetrahedron, resulting in approximations.

Wylie *et al.* [Wyl02a] developed a GPU-based implementation of the projected tetrahedra method. They developed a vertex program algorithm which executed the triangle set determination step of the original PT method. As a vertex program performs the same computations on every vertex, they used a fixed topology graph, mapped to each tetrahedra and corresponding to a triangle fan primitive. By using a look-up table, their algorithm then determines the correct projection class, generating zero-area triangles in certain projection cases. Using a triangle fan allowed to transmit only the data relevant to the four tetrahedral vertices, allowing to render a 1000k tetrahedra mesh at 500k tetrahedra per second with a GeForce 4 GPU. However, they used an approximation to compute the thick vertex color.

Weiler *et al.* [Wei02a] developed a ray-casting based rendering method of individual tetrahedra on the GPU, performing projection in a view-independent way. They used a vertex program to compute the ray-segment exit intersection parameter with each face plane at each vertex. The intersection parameter and vertex scalar values are then linearly interpolated by the graphics hardware to provide at each fragment the coordinates addressing a pre-computed volume integral texture. They were able to render a 220k mesh at 480k tetrahedra/s, on an nVidia GeForce 4. However, computing ray intersections per-vertex instead of at each fragment (due to GPU limitations) resulted in artifacts, especially along tetrahedra edges.

More recently, Kraus *et al.* [Kra04a] reviewed the major causes of artifacts in the PT method and related algorithms. They identified incorrect ray segment length perspective interpolation as being one of the main causes of artifacts and implemented the correct interpolation method, using vertex and fragment processing programs. They also identified linear mapping between ray length and the third

coordinate of the pre-integrated transfer function texture as a source of artifacts, due to an insufficiently accurate sampling of the volume integral for small ray lengths. Using a logarithmic mapping allowed them to reduce the sampling interval for small ray lengths, eliminating edge artifacts due to the use of linear mapping. Coupling this to the use of floating-point alpha-blending virtually eliminated all major sources of rendering errors.

The previously described methods require an object-space visibility sorting. Several object-space sorting methods have been developed, the best-known being the adjacency graph sorting method MPVO (Meshed Polyhedra Ordering Visibility) [Wil92a], using either a Depth-First Search or Breadth-First Search algorithm. Cook *et al.* [Coo04a] improved MPVO, allowing it to generate an image-space correct visibility ordering. For each different graph connected component, their method executes the MPVO topology ordering. Then, it rasterizes every boundary face on the CPU, storing the coordinates of every boundary face fragment into an A-buffer. This allows to define new adjacency relationships, between two consecutive fragments in a pixel list, belonging to the boundary faces of two different cells. These adjacency relationships are then used to extend the MPVO adjacency graph. The authors showed that a depth-first search of this extended graph generates an image-space correct visibility ordering. Adjacency graph sorting methods are typically executed sequentially on the CPU prior to rendering, which can be costly, especially for large datasets. Reducing this cost might increase the overall rendering performance.

Weiler *et al.* [Wei03a] implemented ray-casting of a convex tetrahedral mesh entirely on the GPU, by using fragment processing programs. Using floating-point textures, they were able to store the whole mesh (vertices, face normals and connectivity) in graphics memory, after convexification during a pre-processing phase. They implemented a multi-pass raycasting rendering method. During a given pass, each ray traverses a single cell, accumulates the cell contributions into the framebuffer and proceeds to the exit point adjacent cell. They were able to render between 500k and 600k tetrahedra/s, on an ATI Radeon 9700 graphics card with 128 MB of memory. However, the maximum size of the mesh was limited by the graphics memory size (up to 600k tetrahedra with a 512^2 frame buffer).

More recently, Bernardon *et al.* [Ber04a] improved Weiler *et al.*'s approach by using a depth-peeling technique in order to correctly render non-convex meshes, eliminating the need to perform a costly convexification pass. Their technique extracts successive boundary faces layers, starting the raycasting phase afresh from the currently extracted layer. Furthermore, using a static screen tiling scheme, they were able to reduce the number of raycasting passes. They reported to be able to render up to 1.3 Mtetrahedra/s on a 187 ktetrahedra non-convex dataset.

Callahan *et al.* [Cal04a] developed a hybrid image-space/object-space method, performing a coarse per-primitive sorting step on the CPU then a subsequent per-fragment refined step on the GPU. They used the multiple output buffer capability of modern GPUs in order to implement a fixed-depth sorting network, storing, for each pixel, an unsorted depth sequence of up to 4 fragments. Fragments determined to be the closest to the viewpoint are used to compute the contributions of a corresponding ray segment. They reported to be able to render between 1 and 2 million tetrahedra per second. However, in the case where the difference of the submitted fragment unsorted order and the correct one exceeds 4, an incorrect visibility order is determined and artifacts will appear, which can be frequent for unstructured data sets, where cells can vary greatly in size and shape.

Weiler *et al.* [Wei04a], improving on their previous work, were able to store unstructured meshes into graphics memory in a compressed form using tetrahedral strips and pre-rendering stripification algorithms. Furthermore, using a depth peeling technique akin to the one described in [Ber04a], they were able to correctly render non-convex meshes. They reported to be able to store up to 17 million tetrahedra on a 256 MB graphics card, and up to 3 million with speed optimizations.

3. TETRAHEDRA PROJECTION WITH POINT SPRITES

As we saw earlier, most tetrahedron projection implementations describe each tetrahedron by a set of four triangles, which is required by graphics APIs, which are not designed for the rendering of unstructured volume primitives. As a consequence, for a given tetrahedron, there is an overall duplication factor of up to 3 concerning per-vertex data (vertex coordinates, scalar field value) and per-face data (e.g. face plane equations). As the graphics port downstream bandwidth is limited (approximately 2.1 GB/s for AGP8x), this can result in a potential bottleneck and decrease the rendering performance. Ideally, for a given tetrahedron, its associated data should be transmitted without any duplication.

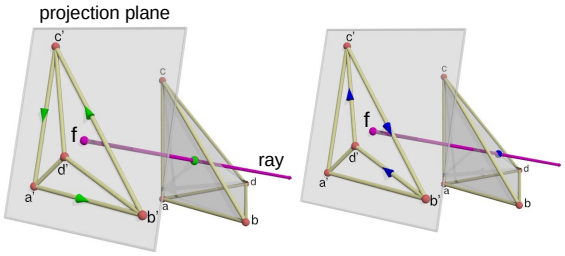
Here, we propose an enhancement to GPU-based tetrahedra projection methods allowing to transmit only the required data. To each tetrahedron, we associate a point sprite primitive instead of four triangles. For each of a given tetrahedron's four vertices, we specify the vertex (x, y, z) coordinates and scalar field value as point sprite vertex attributes, amounting to a total of 16 floating-point values per tetrahedron.

To each rasterized fragment of the sprite, we associate a ray. We use a vertex processing program in order to perform per-tetrahedron constant computations, such as edge line equations, whereas we use a fragment program to compute ray-tetrahedron intersection point depths, using the intermediary results computed by the vertex stage.

As a point sprite has fixed maximum dimensions, we

assume for the remainder of this section that the tetrahedron projection's axis-aligned bounding box is included within the point sprite footprint. Also note that we consider only the case of orthographic projections.

The two vertex and fragment processing programs describe each tetrahedron as a set of four faces, each face being represented by a set of three co-edges, connected into a loop. For any given front-face (in world space), its projection's co-edges normals point towards the center of the face, whereas for any given back-facing face, the normals point towards the exterior of the face. Using this property, we can determine for any fragment whether it belongs to a given back-face or front-face, by computing fragment position-face edge equations dot products. We can then determine the two faces (one front-facing and one back-facing) whose projections cover the fragment, giving the ray entry and exit points. Figure 1 shows how intersected faces are determined.



- f is left to a'b', b'c' and c'a'
- abc is intersected and is a front face
- f is right to b'd', d'c' and c'b'
- bdc is intersected and is a back face

Figure 1. Intersecting faces determination

The pseudo-code below describes vertex stage computations:

- Transform the tetrahedron four vertices into window space
- Compute the point sprite bounding box dimensions
- Compute the line equations (in window coordinates) of each co-edge pair (corresponding to each projected tetrahedron edge)
- For each face vertex, compute the reciprocal of the result of its opposite co-edge line equation applied to the vertex

This amounts to the computation of six line equations, and twelve reciprocals (3 for each of the 4 faces). The second co-edge equations are deduced from the first ones simply by changing their signs. As the result of these computations is constant for any fragment, they can be done during the vertex processing stage instead of the fragment stage. The computed values are then written to output.

The following pseudo-code describes fragment stage computations:

- Compute the unnormalized window-space 2D distance of the fragment to each of the twelve co-edges
- For each of the 4 projected faces:
    - Compute the interpolated fragment depth
    - Compute the interpolated scalar value
    - Determine if the fragment is inside the projected face and whether it is a front-face or a back-face
- Determine the respective identities of the front-face and the back-face (if any)
- Compute the ray-segment length, subtracting the entry face z coordinate from the exit face's one
- Determine color and transparency values using the ray-segment length and write them to output
- If no intersected faces are found, write color and transparency values (0, 0, 0, 1) to output

Normalized Barycentric (NB) coordinates are computed, as illustrated in Figure 2, in order to determine the interpolated z and scalar values at the entry and exit points.

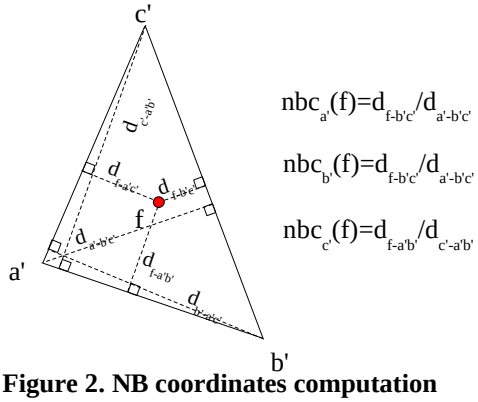


Figure 2. NB coordinates computation

The unnormalized distance from the fragment position to each edge e is computed. It is then multiplied by the reciprocal of the relevant edge-opposite vertex v distance to produce a normalized barycentric coordinate $nbc_v(f)$. NB coordinates are then used to compute the interpolated depth and scalar values at the fragment: $z_{int} = nbc_{a'} z_{a'} + nbc_{b'} z_{b'} + nbc_{c'} z_{c'}$, where $nbc_{a'}$, $nbc_{b'}$, $nbc_{c'}$ are the NB coordinates, $z_{a'}$, $z_{b'}$, $z_{c'}$, the 3 face vertices window-space z coordinate and z_{int} the fragment interpolated z coordinate. Note that this equation is correct only for an orthographic projection. A perspective-correct interpolation formula should be used with perspective projection in order to get a correct result (see [Seg03a] and [Kra04a]).

We use the same formula in order to interpolate the per-vertex scalar values across the entry and exit faces: $s_{int} = nbc_{a'} s_{a'} + nbc_{b'} s_{b'} + nbc_{c'} s_{c'}$. The two scalar values at segment extremities and the segment length are used as a 3D texture coordinate to perform a lookup into a pre-integrated volume integral texture, giving the color and transparency contributions of the ray segment.

4. GPU-ENHANCED BREADTH-FIRST SEARCH

During a given breadth-first search pass, the successor determination of currently examined nodes is performed sequentially on the CPU, only one node's neighborhood being explored at the same

time. However, as each node examination can be performed independently from the others, it is well suited to a parallel implementation, especially on modern GPUs, which are able to process many independent data elements at the same time.

We present a multi-pass CPU-GPU hybrid breadth-first search implementation, executing successor determination on the GPU and subsequently determining visited nodes on the CPU, using the previously GPU-computed successor list. A pre-search phase on the GPU determines the adjacency of each graph node.

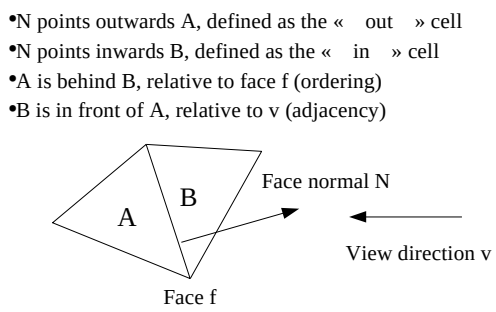


Figure 3. Ordering, adjacency and successor determination

An adjacency graph is an oriented graph defined as {N, E}, respectively a set of nodes and oriented edges, associated to an unstructured mesh. To each mesh cell corresponds a node in the node list N. Two adjacent cells share a common face f which is used to define an ordering relationship, relatively to the face normal vector N, as depicted in Figure 3.

This ordering relationship allows us to compute an adjacency relationship, that is, to determine which one of the two adjacent cells is in front of the other one, relatively to the viewing direction v. The whole set of adjacency relationships is required to compute a correct depth order of the mesh by a graph search. To each adjacency relationship corresponds an edge e in the graph edge list. Note that the adjacency graph must be determined at each view parameter change. The edge list E is determined using the node and mesh faces lists.

Adjacency determination phase

Each list is stored into graphics memory as two different floating point 2D textures (RGBA and luminance). For each list, the dimensions of the two textures are the same. A node list element i stores the node's 4 face indices f, addressing the face list, and the node identifier. A face list element j stores the face normal vector N^j and the indices of the two "in" and "out" nodes sharing the faces.

The adjacency information is determined by the fragment program described below, using the four textures as an input. For each graph node, the program determines its successor nodes and its number of entering edges (in-degree), relative to the viewing direction and writes them to output. A full-screen quadrilateral is rendered, each fragment rasterized corresponding to a single graph node i. The following pseudo-code describes the operations performed:

- Set entering edges number to zero
- Fetch the node face indices and identifier from textures 1 and 2
- Translate the 4 1D face indices f1D into 4 sets of 2D texture coordinates f2D
- For each face f
  - Fetch f's information from face list using f2D
  - Determine the successor node (if any) corresponding to face f relative to the viewing direction
  - If f is not a boundary face, add one to the total entering edges number
- Write successor information (cell indices) to first output color
- Write number of entering edges to 2nd output color

Figure 3 describes in greater detail the determination of successor nodes through a given face f.

After the end of this phase, we copy back the computed successor information from the first output buffer into a floating-point texture, the adjacency texture, that will be used as an input during the search phase, whereas the entering edges information is read back from the second output buffer into main memory.

Breadth-First Search phase

During the search, the graph node list is stored in main memory, storing each node's state (unvisited or visited) and number of entering unvisited edges. The list of nodes currently examined is stored into a list on the GPU, which also maintains a search buffer. The pseudo-code below gives an outline of the search algorithm:

- Compute unvisited nodes count
- While unvisited nodes count greater than zero
  - Perform a search pass
  - Subtract visited nodes count from unvisited nodes count

Source nodes (nodes having no entering edges) are initially set as visited whereas all the other graph nodes are initially unvisited. They are also put into the list of currently examined nodes and subtracted from the unvisited nodes count. The following pseudo-code describes the execution of a given pass, and the operations performed for the two GPU and CPU steps:

For each given pass:

GPU:

- For each currently examined node (fragment)
  - Read its identifier, translate it into 2D coordinates

id2D
    - Use id2D to read the four successor nodes IDs from the adjacency texture
    - Write them to output

CPU:

- Read back the search buffer
- For each node found, decrease the number of unvisited entering edges in the main list
    - If number equals 0
        - Mark node as visited with the pass number
        - Push node into the sorted list of nodes
        - Push node into the list of the nodes to be examined during the next pass
- Send the list of nodes to be examined back to the GPU

The GPU successor determination phase is implemented by a fragment processing program (as outlined by the pseudo-code above). During this phase, we render a quadrilateral such as each fragment rasterized corresponds to a node in the currently examined node list. Successor information, read from the previously computed adjacency texture, is written to output into the search buffer.

After the GPU phase, a sequential search of the readback search buffer is done on the CPU, examining each successor node and updating the list of nodes to be examined during the next pass. This list is then transmitted back to the GPU by updating the corresponding texture.

When the search main loop is over, the sorted identifier list is used as an element buffer in order to specify the mesh cells in the computed depth order.

5. RESULTS

We present results of our breadth-first search implementation and then of our point sprite-based tetrahedron rendering method. We performed measurements on an Intel 3.0 Ghz Xeon-based workstation with 1 GB RAM, an Nvdia NV40-based GeForce 6800 GT graphics board with 256 MB RAM on an AGP8x graphics port, and Redhat Linux Enterprise 4 as an operating system. We used the 1.0-7664 version of the Nvidia Linux graphics drivers. The OpenGL library was used for all our implementations.

Breadth-first search

We implemented the GPU part of our GPU-CPU BFS method with the ARB_fragment_program OpenGL extension assembly language, using specific instructions made available by the NV_fragment_program2 OpenGL extension, such as conditional execution or return instructions. In order to evaluate potential gains, we also made a software implementation of the breadth-first search algorithm, including the adjacency determination method and the actual search algorithm.

To test our implementations, we generated a simple dataset corresponding to a regular hexahedra grid that we converted into tetrahedra as described in [Shi90a], each tetrahedron corresponding to a single graph node. We used several grids of increasing size. For each grid, we performed a sequence of several searches, alternatively specifying two different viewing direction vectors. Table 1 gives adjacency computation and search times as well as the number of sorted nodes per second as a function of grid size, for the CPU-GPU and software-based methods.

Grid size	**40^3**	**50^3**	**59^3**	**70^3**
Nodes	0.32×10^6	0.63×10^6	1.03×10^6	1.72×10^6
CPU-GPU search results				
$t_{adjacency}$ (s)	0.05	0.08	0.11	0.95
t_{search} (s)	0.05	0.09	0.14	0.25
Sorted nodes/s	2.6×10^6	2.8×10^6	3.4×10^6	1.3×10^6
Software search results				
$t_{adjacency}$ (s)	0.04	0.08	0.13	0.22
t_{search} (s)	0.02	0.05	0.11	0.23
Sorted nodes/s	5.1×10^6	4.6×10^6	4.1×10^6	3.7×10^6

Table 1. Graph search performance

We remark that the CPU-GPU search performance, in sorted nodes/s, increases as the grid size grows. However, for a grid size of 70^3, it decreases dramatically to 1.3×10^6 nodes/s. This might be due to the size of the data textures, consuming nearly all the available graphics memory. On the contrary, software search performance decreases as the grid size grows, as for each pass more nodes have to be examined, still in a sequential way, therefore increasing the quantity of work to do.

Point-sprite Based Tetrahedron Rendering

We used the Cg graphics programming language in order to implement the algorithms described in 3, using an FP40 rendering profile.

In order to perform the object-space pre-rendering cell sort, we used the software breadth-first search implementation we mentioned above. Rendering was made using immediate mode, specifying each tetrahedron with 1 glVertexAttrib3f call for each of the 4 vertices (x, y, z) coordinates sets and one glVertexAttrib4f call for the tetrahedron's 4 per-vertex scalar values.

The dataset used for our tests, a tetrahedrized rectilinear grid, stems from a numerical simulation of the propagation of seismic waves. We used the magnitude of displacement vector as the scalar field. Volume rendering reveals the spatial structure of wave amplitude (Figure 4). For each single measurement, we rendered a sequence of 100

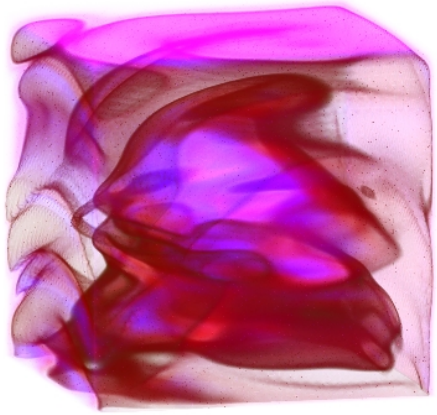


Figure 4. 1250k tetrahedra dataset, 512x512 pixels

images, rotating the view point rotation about the center of the dataset. We measured the average sorting (including adjacency determination), rendering, and total frame time, the latter being used to compute rendering performance. We activated linear filtering of the pre-integrated transfer function texture, which was computed separately by an offline process. The number of tetrahedra rendered per second was measured as a function of resolution.

Table 2 represents execution times of respectively the cell sorting step and the rendering step, the frame total time, and the number of tetrahedra rendered in the 10^6 unit per second.

Resolution	**256^2**	**512^2**	**768^2**	**1024^2**
$t_{sorting}$ (s)	0.33	0.33	0.33	0.33
$t_{rendering}$ (s)	0.36	0.86	1.74	2.92
t_{frame} (s)	0.7	1.21	2.09	3.29
Tetra/s ($x10^6$)	1.78	1.03	0.6	0.38

Table 2. Rendering speed as a function of resolution

We observe that for a resolution of 512^2 pixels, rendering speed is about $1.0x10^6$ tetrahedra rendered/s. Sorting time is constant whatever the resolution, at it is only dependent on the mesh size and viewing direction.

We performed measurements in order to highlight the cost of fragment processing, disabling the sorting step and using a null fragment program doing no computations, only writing the color (1, 1, 1, 1) to output. Table 3 indicates the average rendering time and speed as a function of resolution. We also indicate the ratio between the rendering time shown in Table 2 and the rendering time with no fragment processing.

We observe that for a resolution of 256^2 pixels, rendering speed is about $6.3x10^6$ tetrahedra/s. Rendering speed decreases with resolution whereas the rendering time ratio increases, only slightly decreasing from 512^2 to 768^2.

Resolution	**256^2**	**512^2**	**768^2**	**1024^2**
$t_{rendering}$ (s)	0.2	0.2	0.2	0.27
Tetra/s($x10^6$)	6.27	6.31	6.14	4.64
Rendering time ratio	1.8	4.3	8.7	10.8

Table 3. Rendering speed with no fragment processing

The fact that the time ratio is high, even for low resolutions, indicates that the fragment processing cost seems to be the bottleneck of our rendering method. The increase of the rendering time is probably due to the increasing rasterization costs (including alpha-blending).

Finally, in order to evaluate the bandwidth used to transmit the whole mesh from main memory to the graphics card, we used a null vertex processing program, keeping only the vertex projection and sprite size computation. We also disabled sorting. Table 4 represents rendering time as a function of resolution.

Resolution	**256^2**	**512^2**	**768^2**	**1024^2**
$t_{rendering}$ (s)	0.2	0.2	0.19	0.2
t_{frame} (s)	0.2	0.2	0.2	0.2
Bandwidth (MB/s)	405	398	408	399

Table 4. Bandwidth with no vertex processing

We compute bandwidth as the amount of data transmitted over the rendering time, that is, the ratio between mesh size times tetrahedron size (64 bytes) and the average frame time. A glFinish command ensures that all data be transmitted between the start and the end of the rendering. Bandwidth does not vary with resolution, as fragment processing is disabled, and is lesser than $1/4^{th}$ of the AGP8X theoretical maximum bandwidth. However, rendering time with no vertex and fragment processing is lesser than rendering time with processing activated (see Table 2), which seems to indicate that bandwidth is not a limiting factor. Further investigation is necessary in order to see if the fragment processing and data transmission times balance better with larger meshes.

6. DISCUSSION

As we precedently showed, our point-sprite based rendering method significantly decreases the amount of data transmitted through the graphics port with 64 bytes transmitted per tetrahedron. Furthermore, it decreases the number of vertices processed by the vertex stage to only one per tetrahedron, allowing to perform more complex computations. However, it appears to be limited by the fragment stage, as the

quantity of computations done for each fragment is quite important. Computations that are done at each fragment, such as multiplying each face vertex z coordinate and scalar value by the reciprocal value of the distance to the opposite edge could be done in the vertex stage, saving a significant amount of computation. Interpolated scalar values and fragment depths have to be explicitly calculated by the fragment program whereas triangle-based view-independent methods by specifying them at each triangle vertex can determine them by using hardware linear interpolation. Also note that the size of any projected tetrahedron is limited by the point sprite maximum size. Tetrahedra with a larger footprint will be incorrectly processed. However, GPU fragment processing power has dramatically increased in the past few years, whereas graphics port bandwidth has undergone a much slower rate of growth. We think that this will still be the case for at least several years to come. Therefore, it might be likely that our method's performance will scale better with the increasing GPU fragment processing power that the performance of other projection methods which might become limited by graphics port bandwidth. Future software and hardware evolutions might also eliminate the point sprite size limitation. Moreover let us remind that GPU-based raycasting methods such as described in [Wei03a] or [Ber04a] require to store the whole geometry in graphics memory. Tetrahedron projection methods are not subject to this severe limitation since they allow streaming from main memory.

The partly GPU-based BFS method we described takes advantage of the multiple fragment units of GPUs in order to speed up the successor determination. However it performs a readback of successor information into main memory in order to update the global node list, which costs graphics port bandwidth and CPU time. Nevertheless, for a 1.7×10^6 node graph, the search time for the GPU-CPU based method and the purely CPU-based one are about the same, indicating that GPU successor determination might be fast enough to compensate for the read-back and re-send penalty as well as the additional CPU cost. Eliminating the readback and performing the CPU work on the GPU might increase the CPU-GPU BFS performance in a significant way, allowing it to sort large meshes (with more than 2×10^6 cells).

7. CONCLUSION AND FUTURE WORK

After this first implementation of point sprite-based tetrahedron projection method, we intend to test the rendering of larger datasets, with at least 10^6 cells, and compare its performance with other GPU-based tetrahedron projection method. We also plan to improve our partly GPU-based BFS method by eliminating the CPU visited-node determination, using “render to vertex array” capabilities.

8. ACKNOWLEDGMENTS

We would like to thank Jean-Philippe Nominé for his thorough and insightful reviewing of this work, as well as the anonymous reviewers for their useful comments. We also would like to thank Dominique Rodrigues for PRODIF datasets.

9. REFERENCES

[Ber04a] F.F Bernardon, C.A. Pagot, J.L.D. Comba, C.T. Silva, GPU-based Tiled Ray Casting using Depth Peeling, SCI Institute Technical Report, No UUSCI-2004-006, University of Utah, 2004

[Cal04a] S.P. Callahan, M. Ikits, J.L.D. Comba, C.T. Silva, Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering, SCI Institute Technical Report, No UUSCI-2004-003, University of Utah, 2004

[Coo04a] R. Cook, N. Max, C.T. Silva, P.L. Williams, Image-Space Visibility Ordering for Cell Projection Volume Rendering of Unstructured Data,, IEEE Transactions on Visualization and Computer Graphics, Vol 10, N° 6, 2004

[Kin00a] D. King, C.M. Wittenbrink, H.J. Wolter, An Architecture for Interactive Tetrahedral Volume Rendering, HP Labs Technical Report, HPL-2000-121R3, 2000

[Kra04a] M. Kraus, W. Qiao, D.S. Ebert, Projecting Tetrahedra without Rendering Artifacts, Proceedings of IEEE Visualization 2004, pp. 27-34, 2004

[Max95a] N. Max, Optical Models for Direct Volume Rendering, Transactions on Visualization and Computer Graphics, Vol. 1, N° 2, pp. 99-108, 1995

[Mor04a] K. Moreland, E. Angel, A Fast High Accuracy Volume Renderer for Unstructured Data, Proceedings of IEEE Symposium on Volume Visualization and Graphics 2004, pp. 9-16, 2004

[Nvi04a] NVIDIA Corporation, Programming Graphics Hardware, Eurographics 2004, 2004

[Seg03a] M. Segal, K. Akeley, The OpenGL Graphics System: A Specification (Version 1.5), Silicon Graphics, Inc., 2003

[Shi90a] P. Shirley, A. Tuchman, A Polygonal Approximation to Direct Scalar Volume Rendering, Proceedings of San Diego Workshop on Volume Visualization, Computer Graphics, vol. 24, N° 5, pp. 63-70, 1990

[Wei02a] M. Weiler, M. Kraus, T. Ertl, Hardware-Based View-Independent Cell Projection, Proceedings of IEEE Symposium on Volume Visualization 2002, pp. 13-23, 2002

[Wei03a] M. Weiler, M. Kraus, M. Merz, T. Ertl, Hardware-Based Ray Casting for Tetrahedral Meshes, Proceedings of IEEE Visualization 2003 , pp. 333-340, 2003

[Wei04a] M. Weiler, P.N. Mallon, M. Krauss, T. Ertl, Texture-Encoded Tetrahedral Strips, Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics, pp. 71-78, 2004

[Wil92a] P.L. Williams, Visibility-Ordering Meshed Polyhedra, ACM Transactions on Graphics, Vol. 11, N° 2, pp. 103-126, 1992

[Wyl02a] B. Wylie, K. Moreland, L.A. Fisk, P. Crossno, Tetrahedral Projection using Vertex Shaders, Proceedings of IEEE Volume Visualization and Graphics Symposium 2002, pp. 7–12, 2002

Light Field Rendering using Matrix Optics

Lukas Ahrenberg
MPI Informatik
Stuhlsatzenhausweg 85
D-66123 Saarbrücken, Germany
ahrenberg@mpi-inf.mpg.de

Marcus Magnor
Technical University of Braunschweig
Muehlenpfordtstrasse 23
D-38106 Braunschweig, Germany
magnor@mpi-inf.mpg.de

ABSTRACT

This paper presents a light field rendering framework based on matrix optics. Matrix optics, in contrast to intersection-based methods such as ray-tracing, has the advantage that a generic series of optic operators can be combined into a single matrix. This enables us to realize a "virtual optical bench" where different setups can be easily tested. We introduce the theoretical foundation of matrix optics and define a set of operators suitable for light fields. We then discuss a wavelet compression scheme for our light field representation. Finally we introduce a real-time rendering approach based on matrix optics suitable for both uncompressed and compressed light fields.

Keywords
Light field, matrix optics, ray optics, rendering

1 INTRODUCTION

Since the introduction of image based light fields [GGSC96, LH96] a number of different rendering techniques have been suggested. The standard methods today include both image-space and object-space algorithms. The former is based on ray-tracing, ray-casting or view morphing, while the latter typically involves texture mapping. In this paper we investigate the use of matrix optics for light field rendering. In comparison to other light field rendering methods, no ray-plane intersection test has to be performed for every image pixel. Ray-tracing-based methods can be cumbersome if the imaging system involves a series of optical elements such as lenses or material interfaces causing refraction. In this case the imaging method requires tracing a ray to every element which transforms it in some way. Using matrix optics, a set of operators such as light propagation, thin lenses and dielectric interfaces can be represented using matrices, allowing us to model the whole process as a single matrix. This enables us to model a light field under the influence of an arbitrary series of optical operators by performing a linear transform of its elements. Rendering is a special case of this where a camera model is constructed from lens and propagation operators. Our main contribution in this paper is a matrix optics theory for light field modelling. In addition, we show how the operators can be used when rendering directly from a hierarchical wavelet representation of the light field. The implemented framework can be thought of as a "virtual optical bench", a test environment for optical manipulation of light fields.

The paper is organized as follows: Section 2 presents related work in light field research. Section 3 presents the theory of matrix optics, introduces operators for some optical elements and shows our image formation model. In section 4 we construct a framework for light fields using matrix optics. Results are discussed in 5 before we conclude our work in section 6.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

2 RELATED WORK

Several different methods have been proposed for light field rendering in the past few years. Sloan and Hansen have developed a number of methods using ray-light slab intersection tests targeted at parallel architectures [SH99]. In [BBM+01] Buehler et al. present a general method for rendering by blending views from the original light field using a blending field derived from geometrical and view information. Vlasic et al. merge both view-dependent appearance and view-

dependent shape in what they call Opacity Light Fields [VPM+03]. Goldluecke et al. have developed a GPU-based method for dynamic light field rendering using a warping algorithm [GMW02]. The matrix optics method developed in this paper expresses the ray transport from the light field to the image plane as a linear transform. This allows us to easily model the light field under influence of optical elements such as lenses and interfaces.
In addition to a plain light field representation, our framework also has the possibility to handle wavelet compressed light fields. This allows for high compression ratios and efficient storage and rendering. The efficiency for light field encoding has already been reported in several publications; Lalonde and Fournier use wavelets to store light fields in a hierarchical data structure [LF99], while Peter and Straßer introduce a wavelet representation that allows for efficient storage and progressive transmission of light fields [PS01]. In [LSG02] a light field acquisition, compression and representation system based on a hierarchical wavelet structure is presented. Our approach to data representation is similar to the work of Peter and Straßer, while the basic wavelet tree accessing philosophy has ideas in common with the method of Lalonde and Fournier. However, in contrast to both approaches, we perform interactive image reconstruction using matrix optics.

3 MATRIX OPTICS

In this section we introduce a theoretical model for light fields based on matrix optics. We will only briefly touch the foundations of matrix optics here, a more general introduction can be found in [Fow75] and [GB94]. Matrix optics defines linear operators for a number of optical elements as well as propagation of light between planes in space. This gives an elegant way of computing propagation and optical manipulations of light fields. Standard ray-casting or ray-tracing based methods must compute the ray-path between the individual optical elements, while in matrix optics all operators can be combined using matrix multiplication. The model introduced here is an extension to the matrix operators in optics, suitable for light field transformations.
We will start by defining a ray space and a light field on this ray space. Then we will introduce a set of matrix optics operators for modelling optical phenomena on the light field. Finally we construct a camera model and show how an image is formed from the light field.
Let $\Pi \subset \mathbb{R}^3$ be a plane with an associated coordinate system in 3D space

$$\Pi = (\mathbf{o}_\Pi, \mathbf{n}_\Pi, \{\mathbf{e}_1^\Pi, \mathbf{e}_2^\Pi\}), \tag{1}$$

where $\mathbf{o}_\Pi$ and $\mathbf{n}_\Pi$ is the origin and normal of Π, and $\{\mathbf{e}_1^\Pi, \mathbf{e}_2^\Pi\}$ are the vectors spanning the plane.
The ray space on Π consists of all light rays intersecting the plane,

$$\mathcal{R}_\Pi = \mathbb{R}^2 \times \mathbb{R}^2. \tag{2}$$

Thus, a ray passing through Π is described as a point in ray space with homogeneous coordinate

$$\mathbf{r} = [\mathbf{x}, \mathbf{d}, 1]^t \in \mathcal{R}_\Pi. \tag{3}$$

Above $\mathbf{x} = [x_1, x_2]$ denote the plane coordinates in the frame $\{\mathbf{e}_1^\Pi, \mathbf{e}_2^\Pi\}$, and $\mathbf{d} = [d_1, d_2]$ the directional component along Π's basis vectors.
A light field on Π is a mapping

$$L : \mathcal{R}_\Pi \to \mathbb{R}. \tag{4}$$

Given a ray $\mathbf{r}$, the light field L yields the radiance along that ray. We can now define a set of matrix operators for transforming ray space.

3.1 Propagation Operators

A propagation $\mathbf{P} : \mathcal{R}_\Pi \to \mathcal{R}_{\Pi_p}$ means a transformation of the ray space $\mathcal{R}_\Pi$ of plane Π to the ray space $\mathcal{R}_{\Pi_p}$ of some other plane

$$\Pi_p = (\mathbf{o}_{\Pi_p}, \mathbf{n}_{\Pi_p}, \{\mathbf{e}_1^{\Pi_p}, \mathbf{e}_2^{\Pi_p}\}). \tag{5}$$

Any point in $\mathcal{R}_\Pi$ and its image in $\mathcal{R}_{\Pi_p}$ under the propagation $\mathbf{P}$ must correspond to the same ray in world space.
We assume that all propagation takes place in free space, i.e. that there are no occluding objects between Π and Π_p.
Mathematically, let $W_\Pi(\mathbf{r})$ be the world space ray of $\mathbf{r} \in \mathcal{R}_\Pi$, given by the base point $\mathbf{o}_\Pi + [\mathbf{e}_1^\Pi, \mathbf{e}_2^\Pi]\mathbf{x}$ and the direction $\mathbf{n}_\Pi + [\mathbf{e}_1^\Pi, \mathbf{e}_2^\Pi]\mathbf{d}$.
Then

$$\mathbf{P} : \mathcal{R}_\Pi \to \mathcal{R}_{\Pi_p} \tag{6}$$

is a propagation operator if and only if

$$\forall \mathbf{r} \in \mathcal{R}_\Pi : W_\Pi(\mathbf{r}) = W_{\Pi_p}(\mathbf{P}\mathbf{r}). \tag{7}$$

We will now introduce two different propagation operators: the transport operator, which propagates between parallel planes, and the rotation operator which propagates between rotated planes.

3.1.1 The Transport Operator

The transport operator, $T_\mathbf{v}$, propagates the light to a plane parallel to Π offset by some vector, $\mathbf{v} = [v_1, v_2, v_3]^t \in \mathbb{R}^3$

$$\Pi_\mathbf{v} = (\mathbf{o}_\Pi + \mathbf{v}, \mathbf{n}_\Pi, \{e_1^\Pi, e_2^\Pi\}). \tag{8}$$

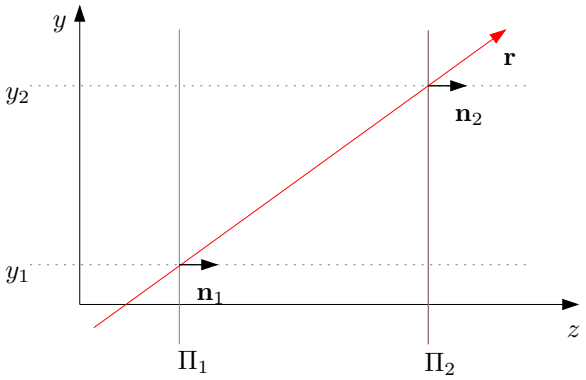


Figure 1: An example of ray propagation. At the plane Π_1, the ray $\mathbf{r}$ has coordinate $\mathbf{r}_{\Pi_1} = [y_1, d]^t$. d is the directional deviation of $\mathbf{r}$ from the normal $\mathbf{n}_1$, and can intuitively be thought of as the slope of r. For an empty space propagation to plane Π_2, the spatial y-component is updated by the transport along the ray direction, while the directional component is unaffected.

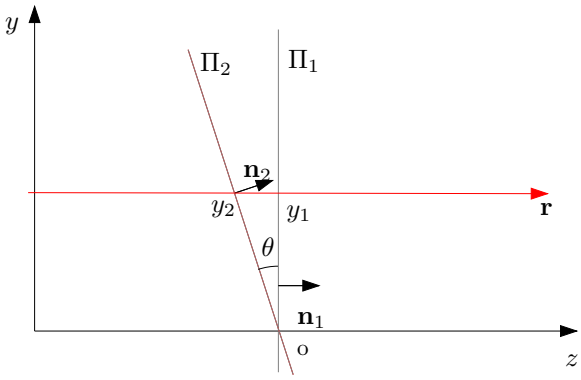


Figure 2: Π_2 is a plane rotated by θ around the origin of Π_1, and $\mathbf{r}$ is a ray traveling in the normal direction of Pi_1, intersecting the planes in y_1 and y_2 respectively. To find the ray coordinate in Π_2, observe that y_2 is scaled proportionally to y_1, given by the triangle $\overline{oy_1y_2}$. As $\mathbf{r}$ intersects Π_2 at an angle of $-\theta$, the direction will be offset by $-\tan\theta$.

It can be written as a 5×5 matrix

$$\mathbf{T_v} = \begin{bmatrix} 1 & 0 & v_3 & 0 & v_1 \\ 0 & 1 & 0 & v_3 & v_2 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{9}$$

which has the desired properties. Intuitively, this corresponds to a transportation along the ray direction and a translation in the plane. Figure 1 shows an example of a pure transportation.

3.1.2 The Rotation Operator

The rotation operator maps the ray space of a plane Π to a ray space of a rotated plane $\Pi_{\mathbf{S}_j^\theta}$, where $\mathbf{S}_j^\theta$ denotes rotation of θ around basis vector $\mathbf{e}_j^\Pi, j \in [1, 2]$:

$$\Pi_{\mathbf{S}_j^\theta} = (\mathbf{o}_\Pi, \mathbf{S}_j^\theta \mathbf{n}_\Pi, \{\mathbf{S}_j^\theta \mathbf{e}_1^\Pi, \mathbf{S}_j^\theta \mathbf{e}_2^\Pi\}). \tag{10}$$

The matrix for ray-space transformation corresponding to the plane rotation of Π around $\mathbf{e}_1^\Pi$ is

$$\mathbf{R}_1^\theta = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1/\cos\theta & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -\tan\theta \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{11}$$

Rotation around $\mathbf{e}_2^\Pi$ follows by symmetry.
A general rotation does not yield a linear operation in ray space. However, it can be adequately approximated if the *paraxial approximation* of ray optics is applied. A linear approximation can be used for rays which lie close to the optical axis. We will use this approximation both for rotations and for elements with curved surfaces in the next section. Figure 2 shows an example where the plane has been rotated around the origin of the plane p_1.

3.2 Lens and Interface Operators

In this section we present operators which map the ray space on Π onto itself.

$$\mathbf{I} : \mathcal{R}_\Pi \rightarrow \mathcal{R}_\Pi. \tag{12}$$

This kind of operator changes the ray direction, and can be used to model elements such as interfaces and thin lenses.

3.2.1 Interfaces

Interfaces are used to model light transition from one medium to another. If the materials have different refraction indices, a perturbation of the ray direction will occur when passing through the material boundary.
The matrix for a planar interface is

$$\mathbf{P}_{n_1,n_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{n_1}{n_2} & 0 & 0 \\ 0 & 0 & 0 & \frac{n_1}{n_2} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

where n_1 and n_2 are the refractive indices of the source and destination materials.
For a circularly curved interface the perturbation of the directional component depends on the plane coordinate component of the ray. The operator matrix is

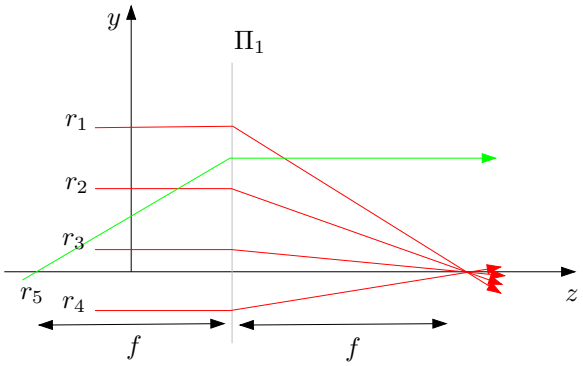


Figure 3: A thin lens in plane Π_1. The rays r_1 to r_4 arrive perpendicular to p_1. Their directions are perturbed depending on the distance from the origin of Π_1 so that all intersect at a distance of f in front of Π_1. r_5 on the other hand, intersects the z-axis a distance of f *in front* of Π_1 and will emanate normal to the plane. A general ray will have its directional component offset by $-y_1/f$.

written as

$$\mathbf{C}_{n_1,n_2,r} = \begin{bmatrix} 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & \\ \frac{1}{r}(\frac{n_1}{n_2}-1) & 0 & \frac{n_1}{n_2} & 0 & \\ 0 & \frac{1}{r}(\frac{n_1}{n_2}-1) & 0 & \frac{n_1}{n_2} & \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (14)$$

As with the planar interface n_1, n_2 denotes the refractive indices, while r is the radius of curvature. A positive r yields a convex interface, while negative values results in a concave one.

3.2.2 The Thin Lens Operator

A lens is considered "thin" if the light propagation within the lens material can be neglected. Thus, the thin lens acts as a perturbator of ray directions, and has the matrix

$$\mathbf{H}_f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1/f & 0 & 1 & 0 & 0 \\ 0 & -1/f & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

for a focal length of f.
The matrix varies the directional component of a light field coordinate as a linear function of the plane component. Figure 3 depicts an intuitive example of a thin lens.

3.3 Image Formation

We will now show how a two dimensional image can be formed from a 4D light field.

Let

$$\Gamma = (\mathbf{o}_{\mathbb{R}^3}, \mathbf{e}_3, \{\mathbf{e}_1, \mathbf{e}_2\}) \quad (16)$$

be the image plane located at the world space origin, where $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ is the standard basis in $\mathbb{R}^3$. To form an image on Γ the intensity of the image plane at a point $\mathbf{x}$ is computed using the general camera model

$$I_\Gamma(\mathbf{x}) = \int_{A_\mathbf{x}} \omega(\mathbf{d}) L(\mathbf{M}[\mathbf{x}, \mathbf{d}, 1]^t) d\mathbf{d}. \quad (17)$$

L is a light field defined on $\mathcal{R}_\pi$. $A_\mathbf{x}$ is the set of all ray directions intersecting $\mathbf{x}$ through the camera aperture, $\mathbf{M}$ is the matrix transforming from $\mathcal{R}_\Gamma$ to $\mathcal{R}_\Pi$ via any optical elements present, and ω is a weighting function used to grade rays dependent on their direction.
However general, this model is computationally expensive. A common practice in real-time computer graphics rendering is to use the pinhole camera model. This is a special case of the general model where the aperture of the camera is considered a single point in space, yielding only a single ray per point in the image plane. If we let the weight $\omega = \delta_0$, so that only rays with $\mathbf{d} = \mathbf{0}$, i.e. perpendicular to the image plane, are considered the camera integral reduces to

$$I_\Gamma(\mathbf{x}) = L(\mathbf{M}[\mathbf{x}, \mathbf{0}, 1]^t). \quad (18)$$

For $\mathbf{M} = \mathbf{1}$ or $\mathbf{M} = \mathbf{T}_\mathbf{v}$ this will render an orthographic view of the light field. Perspective views can simply be rendered by including a lens matrix in $\mathbf{M}$. Thus, a perspective camera with focal length f, viewing a light field from a distance of t, would have the matrix

$$\mathbf{M} = \mathbf{T}_\mathbf{v}\mathbf{H}_f \quad (19)$$

where $\mathbf{v} = [0, 0, t]$.

4 IMPLEMENTATION

In the previous section we have shown how matrix operators can be used to transform light fields. In this section we will discuss our implemented framework.

4.1 Light Field Representation

We have chosen to implement two different ways of representing the light field. The first is a raw light field table, which we will call the *direct* representation, the second is a *wavelet compressed* representation suitable for huge light fields. The former is faster as it basically is a four-dimensional image representation. For each spatial coordinate x_1, x_2, we can look up the RGB value of any ray of direction d_1, d_2. However, as the data size of light fields often gets large, we have also implemented the alternative to use a wavelet compressed representation as described below.

4.1.1 A Wavelet-based Representation

For huge light fields it may be necessary to sacrifice speed and compress the data. We have implemented a wavelet compression scheme where the light field is wavelet-compressed and the coefficients are stored in a space-partioning hexadecary tree structure. This structure is similar to the one presented by similar to the ones presented by Peter and Straßer [PS01]. The data can easily be sent as a progressive stream, making it an attractive choice for data transmission.

Wavelet compression is a well-known approach to data reduction. In the discussion below, we assume the reader to have a basic knowledge of wavelet theory and point to other sources, such as [Dau92] and [SDS96], for a thorough introduction.

Let L be a light field on $\mathcal{R}_\Pi$, and B_i a set of wavelet basis functions. L can then be written as a linear combination of the basis functions

$$L = \sum_{i=0}^{N-1} c_i B_i \tag{20}$$

where N is the total sampled resolution of L, i.e. the number of wavelet basis functions. c_i is called the wavelet coefficients, and for basis functions with good interpolating properties many of these can be dropped or quantized without introducing major visual errors. Thus, lossy compression can be achieved by only storing the important coefficients.

In our wavelet representation of light fields we have made use of two important properties of the basis functions: that they have local support, and space subdivision. A wavelet basis function has local support and is used to refine the value of a basis function on a coarser scale. That is, for a basis function, B_k, of support $s > 0$, one can find basis functions, B_j, $j < k$, of a coarser scale $t > s$, so that their support contains the support of B_k. The support of the basis functions divide the original support of the data set into sub-sets. The basis functions form a space partitioning tree analogous to binary-, quad- and octrees in 1D, 2D and 3D. In analogy we will call this a *hexadeca-tree*, as the children subdivide a parent's support into 16 regions. Using this tree we let each node represent all basis functions of a specific scale and translation. The child-nodes will be those basis functions refining the value along their parent-node's support, subdividing them. In the 1D case of the situation, a binary tree is formed; each node contains only one basis function and the corresponding coefficient, but higher-dimensional trees will have more functions of the same support which are stored in the same tree node. In our 4D case, there will be 16 different basis functions defined having the same support. These will be represented in the same node.

To keep the memory size at a minimum, the node structure is dynamic, storing only the non-zero coefficients and the existing children. We can reduce the memory usage of the hexadeca-tree even more by pruning the tree in a bottom-up approach after compressing the coefficients. As no leaf-node with zero-coefficients will contribute to the reconstructed signal, it can be removed. If all children of a node are removed, it becomes a leaf and the same test can be applied again until we find a node that can not be removed.

The nodes are stored breadth-first in an array. This facilitates progressive decoding so that time-, transmission- or memory-critical applications need only read and decode a part of the tree to obtain approximate rendering results. The approach is similar to the spatial orientation trees in the SPIHT codec for images [SP96].

4.2 Rendering

As seen in Section 3.3, Eq. (18) can be used to render an image from a light field, L. Our framework implements this equation, and computes the image formation matrix $\mathbf{M}$ by having the user specifying a chain of optical elements.

For the direct light field representation, a value lookup is straightforward, and Eq. (18) can be implemented directly. However for the wavelet-compressed representation, the light field must be reconstructed before it can be evaluated. Instead of reconstructing the full light field we have developed a an access method that takes the hierarchical structure of the wavelet tree into consideration. This method is similar to the the work of Lalonde and Fournier [LF99], but integrates it with our operator framework.

Given some image formation matrix $\mathbf{M}$, and an image plane coordinate $\mathbf{u} \in \Gamma$, let $\mathbf{v} = \mathbf{Mu}$. Observe that if $\mathbf{v}$ is located in the support of a specific node in the hexadeca-tree it is bound to be located in the support of one of that nodes children. As the wavelet functions have a value of 0 outside their support, the only nodes that will affect $L(\mathbf{v})$ are the single parent-child chain of nodes whose support contains $\mathbf{v}$. Thus,

$$L(\mathbf{v}) = \sum_{i \in \Omega} c_i B_i(\mathbf{v}), \tag{21}$$

where Ω is the set of all indices of basis functions B with support containing $\mathbf{v}$.

From (18) and (21) we then have the reconstruction expression

$$I_\Gamma(\mathbf{x}) = \sum_{i \in \Omega} c_i B_i(\mathbf{M}[\mathbf{x}, \mathbf{0}, 1]^t). \tag{22}$$

As the children of a node sub-divide the support, it is simple to compute Ω from the root node. Given that a

node contains **v** it is only necessary to check in which of the node's children the point lies until a leaf node is reached. The reconstruction sum will thus take the form of a traversal through the space partitioning tree. This is depicted in Fig. 4.

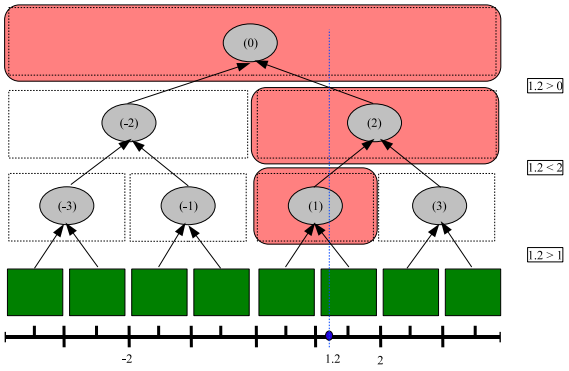


Figure 4: Traversal of a basis node tree, to reconstruct the value at position 1.2. The method starts at the root node, and checks in which child the coordinate lies. Nodes visited in the traversal are highlighted.

We know that the support of the next basis functions in the sum will be contained within the support of the current. The number of summations needed to reconstruct a value in a direct approach will thus be the depth of the tree, which is logarithmically dependent on the resolution of the data. This straightforward method can be directly implemented as an image-space algorithm.

4.3 Antialiasing

A common problem when reconstructing novel views from a light field is aliasing artifacts due to undersampling. This is usually resolved with some kind of interpolation. However, given the 4 degrees of freedom in a normal light field, direct interpolation would be computationally expensive. As most light fields tend to have a higher sampling rate in the spatial dimensions, we use a nearest neighbor interpolation there while performing bi-linear filtering in the directional dimensions, as they tend to have a lower sampling rate, and thus be more prone to aliasing.

5 RESULTS

We have implemented a software framework of our matrix optics representation of light fields. If required, the light field can be stored in a progressive, wavelet compressed data structure. The software renderer computes the current view in a texture and uses OpenGL to map it onto a polygon filling the screen.
For testing we have used both a synthetic light field and the 'buddha4' data set freely available from the Stanford light fields archive [Arc05]. The synthetic data set have a resolution of $128^2 \times 64^2$ samples, each point on the sampling plane covering an angular region of 120×120 degrees. This has proven to be a good balance between spatial and directional resolution while still keeping the original data size manageable. The buddha data set has a resolution of $256^2 \times 32^2$ samples.
The framework lets us implement and test a range of different optical setups. The most interesting application in computer graphics is of course to construct a 'camera' that lets the user interactively view a light field. Figure 5 shows a set of views from one of our synthetic light field. Figure 6 shows four images rendered from the buddha data set. The camera was constructed using two thin lenses operators offset by propagation operators. The camera transform was modeled by a rotation and yet a propagation operator. Aside from light field viewing, we also believe that the availability of other operators, such as interfaces, will allow for easy testing of a range of optical configurations.
We have performed renderings from both a direct and a wavelet compressed representation of the test data set. Rendering speeds are presented in Table 1. The machine used is a Linux-PC with an Intel Xenon 2.8GHz CPU and 2 Gigabytes of RAM.

	No AA	**Bi-Linear AA**
Direct	417 fps	228 fps
Wavelet	79 fps	21 fps

Table 1: Frame rates for rendering a $128^2 \times 64^2$ synthetic light field.

As can be seen from Table 1 the rendering speeds for the uncompressed data representation greatly exceeds those of the wavelet compressed data, making it a preferred choice if the whole data set can be fit into main memory. However, many light field data sets are huge, and may require compression. Nevertheless, our rendering algorithm achieves interactive framerates.

6 CONCLUSIONS

We have presented a light field a set of transformations inspired by matrix optics. This allows us to model optical elements such as lenses and interfaces into the image formation process. On its basis, we have implemented a real-time light field rendering framework. The framework can handle uncompressed light fields as well as wavelet compressed The wavelet compression scheme builds a hierarchical representation of the light field, and we have presented a fast way of accessing the data that integrates well with the presented transformations.
We believe matrix optics proves an elegant solution to modeling optical phenomena for light field rendering, and should provide an interesting complement to

intersection-based methods. The ability to freely combine the operators of different optical elements into one single matrix, results in a lot of flexibility to testing. It should also be noted that the framework is not restricted to pure image-based rendering. Many computer graphics problems can be posed as a sampling or transport of a light field. In such situations this framework can be used to model mappings of light fields through optical elements.

We plan to continue our work by investigating a range of questions and future possibilities. The core work of this paper, the matrix optics operators, is quite general. However, we wish to examine how well they behave for non-linear properties. In addition, aperture stops and ray integration will be implemented via Eq. 17 to allow for effects such as depth of field. It would also be interesting to see if the matrix operators presented here could be used in Fourier Slice Photography as presented in [Ng05].

7 ACKNOWLEDGEMENTS

This work is supported by the EC within FP6 under Grant 511568 with the acronym 3DTV.

References

[Arc05] The Stanford Light Fields Archive. http://graphics.stanford.edu/software/lightpack/lifs.html. World Wide Web, December 2005.

[BBM+01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432. ACM Press, 2001.

[Dau92] Ingrid Daubechies. *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, 1992.

[Fow75] Grant R. Fowles. *Introduction to Modern Optics*, chapter 10. Dover Publications, second edition, 1975.

[GB94] A. Gerrard and J. M. Burch. *Introduction to Matrix Methods in Optics*. Dover Publications, 1994.

[GGSC96] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *Computer Graphics (SIGGGRAPH'96 Conf. Proc.)*, pages 43–54. ACM SIGGRAPH, August 1996.

[GMW02] Bastian Goldlücke, Marcus Magnor, and Bennett Wilburn. Hardware-accelerated dynamic light field rendering. In G. Greiner, H. Niemann, T. Ertl, B. Girod, and H.-P. Seidel, editors, *Proceedings Vision, Modeling and Visualization VMV 2002*, pages 455–462, Erlangen, Germany, November 2002. aka.

[LF99] Paul Lalonde and Alain Fournier. Interactive rendering of wavelet projected light fields. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 107–114. Morgan Kaufmann Publishers Inc., 1999.

[LH96] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics (SIGGGRAPH'96 Conf. Proc.)*, pages 31–42. ACM SIGGRAPH, August 1996.

[LSG02] Reto Lütolf, Bernt Schiele, and Markus H. Gross. The light field oracle. In *Pacific Conference on Computer Graphics and Applications*, pages 116–126, 2002.

[Ng05] Ren Ng. Fourier slice photography. *ACM Trans. Graph.*, 24(3):735–744, 2005.

[PS01] Ingmar Peter and Wolfgang Straßer. The wavelet stream: Interactive multi resolution light field rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 127–138. Springer-Verlag, 2001.

[SDS96] Eric J. Stollnitz, Tony D. Derose, and David H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., 1996.

[SH99] Peter-Pike Sloan and Charles Hansen. Parallel lumigraph reconstruction. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel visualization and graphics*, pages 7–14. ACM Press, 1999.

[SP96] Amir Said and William A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, 1996.

[VPM+03] Daniel Vlasic, Hanspeter Pfister, Sergey Molinov, Radek Grzeszczuk, and Wojciech Matusik. Opacity light fields: interactive rendering of surface light fields with view-dependent opacity. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 65–74, New York, NY, USA, 2003. ACM Press.

Figure 5: Five views of a test light field as taken by a perspective camera. The camera was constructed by combining lens and propagation operators Placement relative to the light field is controlled by a rotation and a propagation operator.

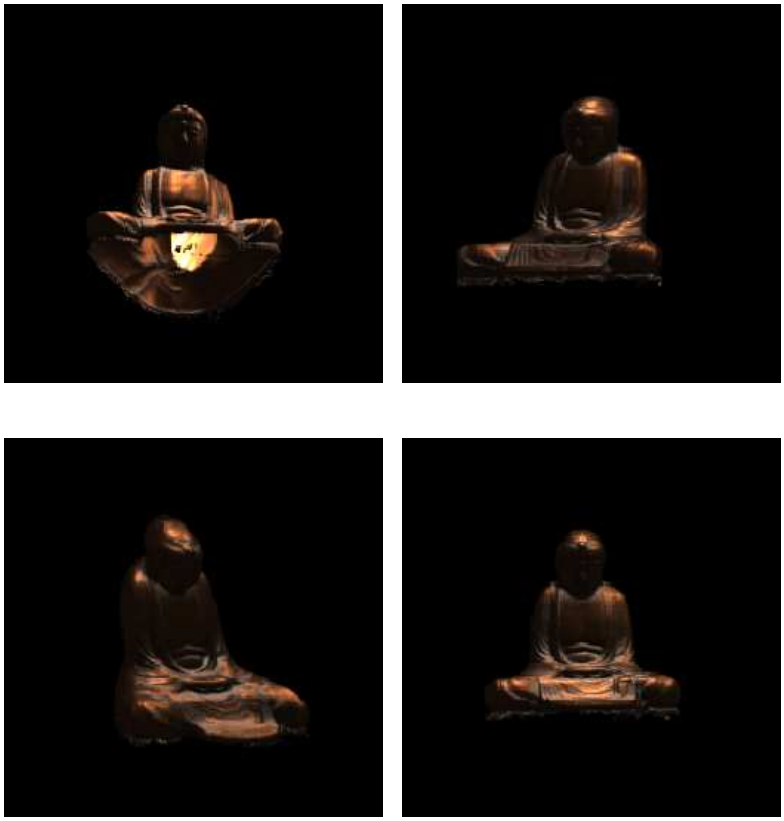


Figure 6: Four views of the buddha light field.

Real Time Simulation of Elastic Latex Hand Puppets

Charles A. Wüthrich[†], Jing Augusto[‡†], Sven Banisch[†], Gordon Wetzstein[†], Przemyslaw Musialski[†], Chrystoph Toll[†], Tobias Hofmann[†]

(†) CoGVis/MMC, Faculty of Media

Bauhaus-University Weimar, D-99421 Weimar, Germany

(‡) ABS-CBN Foundation, Mo. Ignacia St.,

Diliman, Quezon City 1100, Philippines

E-Mail: caw@medien.uni-weimar.de

ABSTRACT

Children television productions have been using puppets for a long time. Since the early days of computer animation, computer puppet simulation has been researched intensively. Complex motion capture equipment allows nowadays the real time mapping of movement for virtual puppets (performance animation). However, the costs of capturing equipment are too high and the difference in the workflow make it difficult for small production teams to access and use such technology. This paper presents a system for the real time simulation of elastic latex hand puppets which are used in television productions. After an analysis of the production processes of real puppets and of the materials used for their production, the paper describes the components of the system simulating them. The system connects a high resolution visual mesh to a three-layered 3D mass spring mesh, which is used for the elastic simulation. Polygonal mesh decimation of the puppet surface model is used as a basis for generating the elastic mesh. From the decimated mesh a new metod is proposed for generating the internal layers of the mass-spring mesh. A data handglove is used for transmitting forces to the elastic mesh, indirectly moving the surface of the virtual puppet in real time. Dataglove interaction maps in a natural way the hand movements of a puppeteer to the computer model. The tradeoffs of the implementation on low cost hardware and its efficiency are also discussed.

Keywords

Real Time Animation, Physical Simulation, Interactive Puppet Simulation

1 Introduction

Children television production has been using puppets for a long time. Shows like the ”Muppet Show” are pleasant remembrances of many people’s youth, and puppet characters are particularly well accepted by the young audience because of their natural sympathy and their flair. Children’s attention is more drawn visually to colourful puppet characters on television than to regular adult actors. When watching a puppet, children consider it one of their own, and relate to the character played by the puppeteer in an easy and sympathetic way. Plenty of puppet characters in television became famous, and were a long term companion of many children’s daily life.

WSCG 2006 proceedings, ISBN 80-86943-03-8

WSCG’2006, January 30 - February 3, 2006, Plzen, Czech Republic. Copyright UNION Agency–Science Press

With the introduction of powerful and affordable PC graphics cards with TV output, and the development of more and more algorithms for the photorealistic rendering of different materials, at least theoretically it is possible nowadays to simulate such puppets in real time and feed the resulting TV output through a normal TV mixer, mixing it into a normal TV production. Modern PCs with high performance graphics hardware have become affordable even for institutions with a relatively low budget, and this makes the exploration of the possibility of using PCs for puppeteering attractive, even for institutions not traditionally computer oriented, such as TV production companies. Since the production of real puppets is extremely time and cost intensive, computer modeling and computer simulation are a feasible alternative to handcrafting. All it takes to control the character is a PC, a good graphics card, and an input device. In theory.

The idea of mapping input devices onto a virtual character is not new: starting from the seminal work of Parke on the acquisition and mapping of faces and ex-

pression onto a computer-animated face [31], already in the late eighties and in the beginning of the nineties, tracked data or data stemming from image acquisition systems was mapped onto movement of animated characters [39, 37, 12], often focussing on facial animation, albeit not in real time. The development of real time motion capture devices allowed sensor driven real time motion mapping onto characters: new devices were developed [15]. Parallel to this, researchers explored how to build appropriate interfaces for interactive physically–based animated characters [23]. One of the major problems tackled was how to map captured movement onto different models in real time, a process which is known as retargeting [7, 19, 38]. Often, the underlying models are skeleton based [30]. A relatively recent good overview of the current state of the art in motion capture mapping is done in [33].

Parallel to this, real time elastic simulation methods were being developed in two dimensions for cloth [26] and in three dimensions for virtual surgery [40, 6, 16]. Such models are based on mass-spring systems, or variations thereof, and are capable of solving ordinary differential equations at interactive frame rates. Surgery systems often require expensive parallel hardware, such as linux clusters, to achieve real time frame rates. Cloth simulation is usually done in two dimensions, since fabric cloth can be adequately approximated by a surface. It models interaction with the body of the character wearing the cloth through external collision forces with the cloth itself.

To the authors' knowledge, an attempt to simulate an interactive, user driven three-dimensional elastic mesh for interactive puppets has not yet been done. The problems arising from such a simulation are multiple: such a simulation is not possible in real time if the size of the underlying elastic simulation mesh is too big. For puppet rendering, instead, a high resolution surface simulation is necessary to render the surface detail. The structure of the surface mesh (necessary for the visualization) has therefore to be detached from the structure of the physical simulation, so that elastic computations can be done in real time, and mechanisms have to be found for "attaching" the physical mesh to the high resolution mesh of the puppet surface modeled. Starting from the modeled surface, mesh reduction algorithms have to be applied to the visualization mesh to generate the underlying physical low resolution mesh. The physical mesh has to be at a sufficient low resolution to allow real time simulation, and has to have multiple layers, at least three, to simulate the physics of the puppet materials.

This paper will present the problems and implementation issues of an affordable puppeteering system which is being developed in an international cooperation between the Bauhaus-University Weimar and the ABS-CBN Foundation, an educational television production institution based in Quezon City, Philippines. The requirements on the system are that it should be low cost [1], that it simulates latex (i.e. elastic) puppets in real time, and that it is coupled through a data handglove to a puppeteer. Latex puppets have been used in the puppetry field for quite some time. Their main advantage lies in their elastic properties: through the combination of a harder latex layer on the exterior and of elastic foam in their interior, they map well the expressive characteristics of a real live being.

Section 2 will give a brief description of real latex puppets. In Section 3 we will provide an overview of the system. Section 4 will propose one method for reducing the complexity of the model and generating a three–dimensional elastic mass–spring mesh so that the physical simulation can be run at a low resolution. Section 5 will explain the detail of the simulation of elastic material, while Section 6 will present how forces are applied to the puppet. Finally, we will draw some conclusions and will outline future work issues.

2 Making a real latex puppet

The production of latex puppets has been discovered a long time ago [4, 9]. However, they have been used mostly for step by step animation purposes [10]. Puppet stepwise movement is achieved through the insertion of stiff but bendable materials such as thick iron wire. The animation is then produced by moving the puppet slightly frame by frame. Step by step animation gives the animator excellent control on the results, but requires long production times.

Recently, television stations used latex puppets interactively for real time television production. The operation of the puppets is quite simple. In puppets having a complete body, "loose" parts, such as arms, legs and ears, are manipulated interactively through rods and ropes attached to the puppet. Main body parts instead are moved through a hand inserted in the foam filling the body of the puppet. The hand inserted in the puppet manipulates the head, the mouth, and, more rarely, its ears. Figure 1 shows how the puppeteers manipulate such a puppet. In this paper, we will concentrate on puppets having a hand inserted into them.

The process of production of a puppet is long and tedious. It involves several drying phases, which are very sensitive to the atmospheric conditions of the environment. Due to the fact that such puppets can vary a lot in size, it is not always possible to put the puppets for drying in a controlled environment. Mistakes in the creation process or temperature changes result often into unusable or malformed puppets. As a consequence, very often two or more identical puppets are produced at the same time.

[1] Children television productions have generally a much lower budget than adult television ones.

Figure 1: Puppets are manipulated by inserting the hand in the head. Loose parts are manipulated by a second puppeteer, eventually with rods.

The structure of such elastic puppets is composed of three layers. Externally, a latex rubber layer constitutes the skin of the puppet. This layer can have different consistencies, depending on the number of layers of latex being applied to the surface. Internally, a mattress similar foam is injected in the puppet. The purpose of the foam is to fill in the puppet so as to transmit the movement of the hand of the puppeteer to the surface of the puppet. Finally, a hole is dug in the foam to allow the puppeteer to insert his hand. Mouth movement can be improved by glueing a bent rubber mat into the mouth cavity, so that when the puppeteer moves the hand, the two lips are moved symmetrically. Puppets are painted and hair and cloth accessories are applied to them to enrich visual detail. Figure 2 shows such a puppet.

Figure 2: Froggy, a latex puppet.

The materials used for the puppet give it a quite stiff consistency. The latex surface gets stiffer the more layers are applied, while the foam provides for an even distribution of the forces generated by the hand of the puppeteer onto a large part of the puppet. The puppet is flexible and stiff at the same time, and elastically returns naturally to its resting position when moved.

3 Overview of the system

For puppet simulation in real time, the system should be able to feed the output directly into the TV mixer through the graphics card. For the input, a low cost dataglove in the 100$ price range (Powerglove P5™by Essential Reality) is used. All simulations were done on a low cost 1.7 GHz P4 PC, with an NVidia FX 5700 graphics card with TV output[2].

The virtual simulation of a puppet can be basically subdivided into three parts: input capture, physical simulation of the elastic materials, and mapping of the results of the physical simulation onto the puppet surface. Once the deformed puppet surface is generated, the new positions of the polygon mesh are passed to the graphic card and visualized by it.

Input capture is done through a device independent abstract layer. Data is read through the dataglove libraries, and prepared for the physical simulation.

The physics simulator implements a three-dimensional mass-spring system. On one side, some nodes of the mass-spring system are attached directly and controlled by the finger positions of the dataglove. On the other side, the external nodes of the mesh are attached to the surface of the virtual puppet. The three-dimensional mass-spring mesh is composed of three different layers. Layers generation from the original model is presented in section 4, and the mass spring system real time simulation in section 5.

When the fingertips move, the corresponding nodes to the mass spring mesh apply forces to their neighbours, and movement is transmitted to the surface of the puppet. The mass–spring simulation computes in real time the resulting new positions of the mesh nodes, and deforms the puppet surface display mesh accordingly. Finally, the newly positioned surface geometry is rendered and displayed.

4 Discrete 3D mesh generation

Our puppet models are created using standard digital content creation tools. To preserve the visual quality along with a robust simulation that works at interactive frame rates, we separate the polygonal surface from the simulation mesh. Custom decimation methods are applied to the puppet model generating a three-dimensional tetrahedral mesh from the decimated surface. This mesh is attached to the original surface and updates its shape after each simulation step.

4.1 Decimating the original model

Mesh simplification and multiresolution data structures have many applications in Computer Graphics. They can be used for collision detection, ”level of detail” (LOD) rendering or physical computations. An extensive comparison of different algorithms can be found in [13]. A ’good’ physical mesh should preserve the shape of the original model as well as possi-

[2] This due to budget restrictions of the commissioning institution.

ble, it should have edges with almost equal length and no side flipped faces.

We use a custom combination of three different mesh decimation algorithms. In all cases, a scalar is assigned to each vertex v indicating the decimated surface error Δv. Such error equals the priority of the vertex of being removed after the current iteration. The vertex with the highest error is deleted and the procedure is repeated until a predefined number of vertices remain or the error of removing more vertices becomes too high.

The first decimation method used is the normal flipping mode, which computes the maximum angle of each vertex normal n_i to its neighboring face normals n_{ij}. The error is equal to the angle and can be determined by the scalar product

$$\Delta_i^n = max\left\{cos^{-1}\left(\frac{n_i \cdot n_{ij}}{|n_i|\,|n_{ij}|}\right)\right\} \qquad (1)$$

In case of unit length normals the denominator is 1 and can be discarded. Normal flipping is used to prevent highly curved regions from being decimated and assure side flipping.

The second method preserves equidistant triangles within the decimated mesh. We define the normalized roundness R^* of a triangle as a metric for its equidistance. The roundness R is the ratio between the circumference's radius and the shortest edge's length. The circumference's radius r can be computed as $r = \frac{a}{2sin(\alpha)}$. The length of the edges of an equidistant triangle with unit radius is $\sqrt{1/3}$, thus $R^* = \frac{\sqrt{1/3}}{R}$.

The actual roundness is computed for each face that would be created if the vertex was removed. The priority of vertex i is the minimum of 1-(roundness of each neighboring face j) $\Delta_i^r = \min\left\{1 - R_j^*\right\}$.

Surface simplification using quadric error metrics was introduced by [17] and later extended [18] to correctly decimate colored and textured meshes. It is currently one of the most common mesh decimation techniques used in many 3D–modelers.

Arbitrary vertex pairs are iteratively contracted $(v_1, v_2) \rightarrow \overline{v}$, incident edges are connected to $\overline{v}$, v_2 as well as all edges and faces which have become degenerated are removed. The surface error at vertex $v = [v_x v_y v_z 1]^T$ is expressed by a symmetric 4x4 quadric matrix Q of the quadratic form $\Delta^q\{v\} = v^T Q v$. For a given contraction $(v_1, v_2) \rightarrow \overline{v}$ a new matrix $\overline{Q}$ is derived using the additive rule $\overline{Q} = Q_1 + Q2$. The position of $\overline{v}$ is determined by minimizing $\Delta\{\overline{v}\}$, which can be done by computing the partial derivatives of

$$\begin{aligned} v^T Q v \quad = \quad & q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + \\ & q_{22}y^2 + 2q_{23}yz + 2q_{24}y + q_{33}y^2 + \\ & 2q_{34}z + q_{44}, \end{aligned}$$

thus

$$\overline{v} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 1 & 1 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad (2)$$

Garlands simplification produces the best visual results of these algorithms, while equidistance of decimated triangles is preserved by taking the roundness into account. Normal flipping prevents faces from flipping direction. We compute the error for each vertex as a combination of these three error metrics. Usually normal flipping only indicates an invalid removal if the maximum angle is higher than a certain threshold $\Delta^n\{v\} \geq \tau$ we have an invalid removal. Roundness and quadric error are combined according to weights which are dependent on the puppet model $\Delta\{v\} = \lambda\Delta^r + (1-\lambda)\Delta^q$.

The high resolution visual polygon mesh is deformed after each simulation step according to a connection to the simulation mesh. We use the correspondence between the decimated and the original surface to create a connection map that stores weights and connections from the surface of our tetrahedral layered mesh to the visual structure. Each surface vertex is connected to the decimated vertex with the minimum edge distance. If more that one decimated point is possible all the vertices are connected. Each connection to vertex v_i from the direct connectors $\{v_j\}$ gets a weight ω_i that is relative to the fraction of the distance to the connected vertex and the sum of all distances to vertices that are connected to v_i:

$$\omega(v_i, v_j) = \frac{\delta(v_i, v_j)}{\sum_j \delta(v_i, \{v_j\})} \qquad (3)$$

Updating v_i according to the simulation is done by translating v_i according to the translation T_j of its connectors with respect to the assigned weights ω_{ij}:

$$T(v_i) = \sum \omega_{ij} T(v_j). \qquad (4)$$

4.2 Internal layer generation

Taking the resulting decimated surface as initial source, we introduce a method to generate a full and closed layer of tetrahedrons underneath it.

The reason of creating layers instead of filling the model with a uniform mesh is that real time animation of mass-spring systems is still a computational expensive task with time complexity of $O(n+e)$, where n is the number of mass-points and e is the number of springs [21]. While simulating structured meshes is more stable and in general faster, such meshes do not fit to complicated geometric shapes well. Since in our case the amount of storage and the preprocessing time does not influence the simulation, we decided to use an

advancing front approach combined with the Delaunay criterion [5] to create the physical structure. The solution we have chosen allows at the same time to access different layers of the mesh and adjust attributes like spring constants and mass values, so that different degrees of stiffness can be reached.

In the first step we use the "biting spheres" approach [24] to provide proper points in the domain. This method is based on the sphere packing: the basic idea is to fill a three-dimensional geometric domain with spheres (bubbles) to generate new points in the interior. Since we want to create only layers of tetrahedrons instead of stuffing the whole object, we combine this method with the advancing front approach as proposed by Li [25].

Starting from the reduced surface mesh we define a sphere on each vertex with the radius of the half of the distance to the next neighbour vertex. On each intersection of at least three spheres we create new points in the interior of the object. Iterating this procedure with the newly created points generates the next layer of steiner points in the geometric domain.

Next, we use the Delaunay criterion to create new tetrahedrons from existing points, which we define as follows: let P be subset of the whole geometric domain Ω where P contains the inner points p_l created by the procedure described above. Assume that none of the faces $\triangle_{p_j p_k p_l}$ containing the points $p_j p_k p_l \in \Omega$ is a surface face. The Delaunay point for every $\triangle_{p_j p_k p_l}$ is a point $p_{delaunay} \in P$ such that:

$$\begin{aligned}(\|p_{delaunay} - p_j\| &< \|p_{delaunay} - p_m\|) \wedge \\ (\|p_{delaunay} - p_k\| &< \|p_{delaunay} - p_m\|) \wedge \\ (\|p_{delaunay} - p_l\| &< \|p_{delaunay} - p_m\|),\end{aligned}$$

where $\forall m \neq j,k,l$ and $p_m \in P$.

In fact, this condition is still not sufficient to find proper points for creating new cells, therefore prior tests such as verifying the positive distance of the points with respect to the faces are needed. This boils down to finding the proper points $p_{delaunay}$, and connecting them with their corresponding faces $\triangle_{p_j p_k p_l}$ creates new tetrahedrons $tet_{p_j p_k p_l p_{delaunay}}$. According to the generated mesh, we create mass-points and springs for the physical system.

To reach best elasticity while preserving shape different stiffness properties of springs and different mass values are needed in the different layers. Since we can separately access springs and mass-points lying on the surface, on the interior, or even in between, changing their attributes ends up in different physical behavior of the simulated material.

During the simulation, forces applied on several points propagate stepwise to the whole structure. To transmit the deformations to the surface, each outer mass-point is tied in position to its reduced surface vertex, and after every simulation step the position of the polygonal surface is updated. After this operation, a separate rendering module can access the data and display the puppet.

5 Real time elastic simulation

The first issue that has to be addressed when implementing latex puppets is the simulation of the physical properties of the materials involved. Since a layered structure of different materials is to be simulated, an adjustable and flexible physical model is needed, which allows that different extents of elasticity, viscosity, plasticity, etc. are present within it. The model must furthermore be able to perform in real time.

Mass–spring systems meet these requirements, and have been frequently used for the simulation of deformable objects, such as hair [32, 1, 2], cloth [27, 14, 21][3] and three dimensional bodies [8, 22]. Terzopoulos was among the first to suggest mass–spring systems in computer graphics [35, 34, 36] for simulating elastic behaviour. In 1988, Miller [28] simulated dynamics of snakes and worms using a chain of masses and springs. With an increase of computation power of computer systems, more complex structures have been animated, and it is nowadays possible to simulate them in real time.

Mass–spring systems involve complex mathematics: they base on Newtons fundamental law $\vec{F} = m\vec{a}$. For a model consisting of n masses a system of n second order differential equations (ODEs)

$$M\ddot{p} = F(t, p, \dot{p}) \tag{5}$$

has to be solved. The n dimensional vector $p = (p_1, p_2, \ldots, p_n)^T$ represents the positions $p_i = (x_i, y_i, z_i)^T$ of all points. The $n \times n$ dimensional matrix M stores the masses of all particles on its diagonal, and F is a function which describes the system's force field at time t. Equation 5 can be reduced to the system of $2n$ ODEs of first order

$$\frac{d}{dt}\begin{pmatrix} p \\ \dot{p} \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} p \\ v \end{pmatrix} = \begin{pmatrix} v \\ M^{-1}F(t) \end{pmatrix} \tag{6}$$

by introducing the velocity $v = (v_1, v_2, \ldots, v_n)^T$ as a separate variable.

There exist several ways to numerically solve such a system of ODEs. Numerical time integration methods range from the simple explicit Euler scheme [28], which is very fast at expense of accuracy and instability, over higher order explicit methods such as Runge Kutta, to implicit predictor–corrector schemes [3, 27], which are considered more stable [11].

[3] See [26] for an up to date overview of deformable models in cloth simulation.

For our system, we use the implicit Euler scheme

$$\begin{aligned} p(t+h) &= p(t) + hv(t+h) \\ v(t+h) &= v(t) + hM^{-1}F(t+h) \end{aligned} \quad (7)$$

to find the positions at the next time step $t+h$. Here, the system's force field $F(t+h)$ has to be approximated through the second order Taylor series

$$F(t+h) = F(t) + J\Delta p(t+h) + H\Delta v(t+h), \quad (8)$$

where J is the Jacobian and H the Hessian matrix of the system's force field with respect to the positions and, respectively, velocities. With $\Delta p(t+h) = hv(t+h)$ and $\Delta v(t+h) = hM^{-1}F(t+h)$ in Equation 7 we derive

$$\Delta p(t+h) = h(v(t) + \Delta v(t+h)), \quad (9)$$

which is a non–linear equation of the form

$$y(x+h) = f(x+h, y(x+h)). \quad (10)$$

Equation 10 can be solved iteratively through

$$\begin{aligned} y(x+h)^0 &= f(x, y(x)) \\ y(x+h)^{\mu+1} &= f(x+h, y(x+h)^\mu), \end{aligned} \quad (11)$$

where μ represents the iteration number. As suggested in [21], we use the result after one iteration which is sufficient approximation of the solution for real time purposes.

In order to imitate the multi–layered material behaviour of real puppets, a three–dimensional mesh has been created as described in Section 4, and layers have been defined. The entire mesh forms a single mass–spring system. The layered structure is achieved by tuning parameters of the mass–spring system according to the desired materials properties of each layer. This results in a mass–spring system in which the layers result from regions having the same parameter values. Material properties can be made to resemble foam, latex or other materials used for real puppets.

The parameters influencing the behaviour of the puppet are:

- The *time step* h, which has a very strong influence on the behaviour of the simulation. It can be set by the user within reasonable limits. A very small time step decreases the system's reactiveness because of increased computations, whereas a too large time step will lead to instability. Once set, the step size usually stays constant over the whole simulation.
- The *mass* M of the physical points, which determines how big the inertia on the system is. If masses are increased, the reactiveness to external and internal forces is reduced. Thus M can be used to achieve different behaviours of the model.
- The *spring constants* C_{spring}, which are used to calculate the Hookean part of the internal forces. A layered structure (e.g. foam and latex) can be modeled by adjusting the spring constants within the layers.
- The *damping coefficients* C_{damp}, which determine how quickly mass points can be moved by spring forces. It is therefore a measure of the viscosity of the deformable body and strongly affects its elastic behaviour.
- The *topology* T of the three–dimensional mesh, which influences the stiffness of the puppet, and is given by the arrangement and density of mesh points within the puppet's body.

Mass–spring systems are often prone to instabilities, especially if an explicit integration scheme is used. The time step h has to be set very small to ensure stability. We use the implicit Euler method because it is accurate while allowing the step size h to be larger [11]. Assuming that the non–linear system of equations is solved properly, the implicit Euler method is unconditionally stable [20] because the systems force field is consistent over time. If the non–linear system of equations is solved iteratively, as in our case, new restrictions on stability arise. A detailed mathematical analysis of the problem is not done here, since the majority of parameter configurations of M, C_{spring}, C_{damp} and T do not cause instabilities when the time step h is within a proper range, i.e., for normal usage of the system.

For puppet simulation real time interaction is required, and the performance of the system is very important. The easiest way to decrease the computational complexity is to reduce the size of the mass–spring system, i.e., to decrease the number of mass points and springs. Table 1 shows the performance of the physical solver. At the required frame rate of a PAL television (25fps) it is possible to simulate in real time a mass–spring system constituted by up to 5000 mass points and 14600 spring links. Through the decimation and layer generation methods presented above, we reduced the models to such sizes.

Masses	Springs	Update time (in s)
3744	1872	0.015
3744	6952	0.025
5664	15056	0.043
9978	25892	0.076

Table 1: Performance of the physical system.

6 Applying forces to the puppet

To generate the shape of the Puppet, a separate modeler is used. Five node regions on the physical mesh are marked by hand, one for each finger of the data handglove. The individual regions must of course consist of adjacent points, and must be disjunct. The marked points are preserved through the mesh reduction process, so that the corresponding nodes do not disappear in the simplification.

When the puppeteer moves his fingers, the movement of the fingertips is converted into forces according to Newton's first law, and fed to the physics engine as external forces acting on the corresponding nodes. The physical simulation then computes the state of the mass–spring system in the next time interval, and updates the position of the mesh nodes. Since puppets are used in live production the system must be tuned for output at TV signal frame rates.

Figure 3 shows the result of opening the dataglove on the face of a computer generated puppet. The accompanying video shows a real time recording of the computer output while operating the puppet.

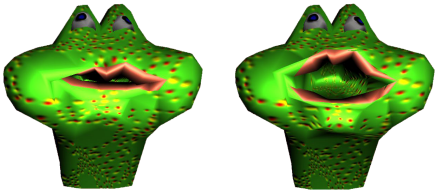


Figure 3: Elastic simulation propagates the movement of the dataglove to the mouth of the puppet

7 Conclusions and future work

We have presented a system for the simulation of hand puppets made of latex. The system is capable of running the simulation in real time on low cost machines, allowing a puppeteer to interactively "operate" the puppet, and allowing real time output feeding in a chroma key device, and consequently the mixing of the simulated puppet with a live television signal.

The system couples a data hand glove to application points of a three-dimensional mass-spring system. When the finger sensors of the data glove are moved, a force is applied to the corresponding points of the mass-spring system, which in turn propagate the movement to the rest of the nodes. The surface of the puppet is attached to the mass-spring system, and therefore is moved indirectly by the hand of the puppeteer. The system is capable of computing in real time (at 25 frames per second) the dynamic simulation for a mass-spring system having more than 5000 mass points and around 14600 springs, and to display the deformed surface of the puppet at the frame rate mentioned above.

Since the resolution of the mass spring system is much lower than the polygon resolution needed to display the detail needed for a broadcast quality puppet, a method for the derivation of the mass-spring mesh from the surface mesh of the puppet has been developed. This method computes first a simplification of the surface mesh, reducing the number of polygons to a predefined number. The centers of the resulting polygons are glued to the surface mesh of the puppet and constitute the first layer of the physical mass spring mesh. From this reduced layer, a new method for the generation of the internal layers of the mass spring system has been developed.

Although the system works well in all tested cases, the resulting mass spring mesh is not uniform. This because polygon simplification algorithms are optimized for visual shape appearance, not for regularity. It is known [29] that uniform tetrahedral meshes behave well as mass-spring systems. Theoretically, this makes the system prone to instability. However, even during several long test session made by children, the mesh topology did not generate unexpected behaviours. We are currently working on mesh simplification algorithms tuned for mesh regularity.

One of the challenging aspects of numerically solving partial differential equations is understanding how parameters like time step or mesh configuration influence the stability of the solvers. Time step size adaption helps greatly the stability of the system at the cost of speed. It would be of benefit to study when and at which costs step adaption can to be applied.

Acknowledgments

We are grateful to the ABS-CBN Foundation, Quezon City, Philippines for allowing Jing Augusto to come to Weimar for the entire project duration. Thanks also to the students that have been involved in the project and implemented parts of the system: Uwe Hahne, Bernhard Bittdorf, Benjamin Schmidt, Annekathrin Linge, Andreas Emmerling, Andreas Kunze, Jonas Schild, Sebastian Knoedel and Marc Pelao.

References

[1] K. Anjyo, Y. Usami, and T. Kurihara. A simple method for extracting the natural beauty of hair. In *Proc. of SIGGRAPH '92*, pages 111–120. ACM Press, 1992.

[2] Y. Bando, T. Nishita, and B. Chen. Animating hair with loosely connected particles. *Comp. Graphics Forum*, 22(3):411–418, 2003.

[3] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proc. of SIGGRAPH '98*, pages 43–54. ACM Press, 1998.

[4] J. Bell. *Strings, Hands, Shadows: A Modern Puppet History*. Detroit Institute of Arts, Detroit, MI, 2000.

[5] M. Bern and P. Plassmann. Mesh generation. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 6. Elsevier, 2000.

[6] D. Bielser, V. Maiwald, and M. Gross. Interactive cuts through 3-dimensional soft tissue. *Comput. Graph. Forum*, 18(3):31–38, 1999.

[7] B. Bodenheimer, C. Rose, S. Rosenthal, and J. Pella. The process of motion capture - dealing with the data. In *Proc. of the Int. Conf. on Computer Simulation and Animation 97, Budapest, Hungary*, pages 3–18. Springer Verlag, Vienna, Austria, 1997.

[8] D. Bourguignon and M.-P. Cani. Controlling anisotropy in mass-spring systems. In *Proc. of Computer Animation and Simulation '00*, pages 113–123, Wien, aug 2000. Springer.

[9] T. Brierton. At last, foam puppet fabrication explained! *Animation World Magazine*, 1(2), 1998.

[10] T. Brierton. *Stop-Motion Puppet Sculpting: A Manual of Foam Injection, Build-Up and Finishing Techniques*. McFarland & Company, Jefferson, N.C., 2004.

[11] I. Bronstein, K. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, Thun, third edition, 1997.

[12] G. Cameron, A. Bustanoby, K. Cope, S. Greenberg, C. Hayes, and O. Ozoux. Motion capture and CG character animation (panel). In *Proc. of SIGGRAPH'97*, pages 442–445, 1997.

[13] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(6):37–54, 1998.

[14] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Proc. of Graphics Interface '99*, pages 1–8, Kingston, ON, June 1999.

[15] C. Esposito, W. B. Paley, and J.-C. Ong. Of mice and monkeys: A specialized input device for virtual body animation. In *Simposium of Interactive 3D Graphics*, pages 109–114, 213, Monterey, CA., 1995.

[16] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Computer Graphics Forum*, 19(3), 2000.

[17] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proc. of SIGGRAPH '97*, pages 209–216. ACM Press, 1997.

[18] M. Garland and P. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Proc. of IEEE Visualization 98*, pages 263–269, 1998.

[19] M. Gleicher. Retargeting motion to new characters. In *Proc. of SIGGRAPH 98*, pages 33–42, 1998.

[20] M. Hauth. *Visual Simulation of Deformable Models*. PhD thesis, Univ. of Tübingen, 2004.

[21] Y.-M. Kang and H.-G. Cho. Complex deformable objects in virtual reality. In *VRST '02: Proc. of the ACM symposium on Virtual reality software and technology*, pages 49–56. ACM Press, 2002.

[22] U. G. Kühnapfel, H. K. Çakmak, and H. Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24(5):671–682, 2000.

[23] J. Laszlo, M. van de Panne, and E. Fiume. Interactive control for physically-based animation. In *Proc. of SIGGRAPH 00*, pages 201–208, 2000.

[24] X.-Y. Li, S.-H. Teng, and A. Üngör. Biting spheres in 3d. In *8th Int. Meshing Roundtable*, pages 85–95, 1999.

[25] X.-Y. Li, S.-H. Teng, and A. Üngör. Biting: advancing front meets sphere packing. *Int. Jour. for Numerical Methods in Engg(2000)*, 2000.

[26] N. Magnenat-Thalmann, F. Cordier, M. Keckeisen, S. Kimmerle, R. Klein, and J. Meseth. Simulation of Clothes for Real-time Applications. In *Proc. of Eurographics 2004, Tutorial 1*, 2004.

[27] M. Meyer, G. Debunne, M. Desbrun, and A. Barr. Interactive animation of cloth-like objects in virtual reality. *Journ. of Visualisation and Comp. Anim.*, 2000.

[28] G. S. P. Miller. The motion dynamics of snakes and worms. In *Proc. of SIGGRAPH '88*, pages 169–173. ACM Press, 1988.

[29] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *Proc. of the 12th Int. Meshing Roundtable*, Santa Fe, NM, 2003.

[30] S. Oore, D. Terzopoulos, and G. Hinton. Local physical models for interactive character animation. *Comp. Graphics Forum*, 21(3):1–17, Sept. 2002.

[31] F. I. Parke. *A Parametric Model for Human Faces*. PhD thesis, Dept. of Computer Science, Univ. of Utah, 1974.

[32] R. E. Rosenblum, W. E. Carlson, and E. Tripp, III. Simulating the structure and dynamics of human hair: modelling, rendering and animation. *Journ. of Visualisation and Comp. Anim.*, 2(4):141–148, 1991.

[33] H.-J. Shin, J. Lee, S.-Y. Shin, and M. Gleicher. Computer puppetry: An importance-based approach. *ACM Trans. on Graphics*, 20(2):67–94, 2001.

[34] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *Proc. of SIGGRAPH '88*, pages 269–278. ACM Press, 1988.

[35] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proc. of SIGGRAPH '87*, pages 205–214. ACM Press, 1987.

[36] D. Terzopoulos, J. Platt, and K. Fleischer. From goop to glop: Heating and melting deformable models. In *Proc. of Graphics Interface '89*, pages 219–226, 1989.

[37] D. Terzopoulos and K. Waters. Analysis and synthesis of facial image sequences using physical and anatomical models. *IEEE Trans. on Patt. Analysis and Mach. Intelligence*, PAMI-15(6):569–579, 1993.

[38] S. Vacchi, G. Civati, D. Marini, and A. Rizzi. Neo euclide: A low-cost system for performance animation and puppetry. In *Proc. of Gesture Workshop '03*, pages 361–368, Wien, 2003. Springer Verlag.

[39] L. Williams. Preformance-driven facial animation. *Proc. of SIGGRAPH '90*, 24(4):235–242, 1990.

[40] R. Yagel, D. Stredney, G. Wiet, P. Schmalbrock, L. B. Rosenberg, D. Sessanna, and Y. Kurzion. Building a virtual environment for endoscopic sinus surgery simulation. *Computers & Graphics*, 20(6):813–823, 1996.

Frequency-Based Representation of 3D Models using Spherical Harmonics

M. MOUSA R. CHAINE S. AKKOUCHE
L.I.R.I.S : Lyon Research Center for Images and Intelligent Information Systems
Bâtiment Nautibus, 8 boulevard Niels Bohr
69622 Villeurbanne Cedex, FRANCE
{mmousa, rchaine, sakkouch}@liris.cnrs.fr

ABSTRACT

3D meshes are the most common representation of 3D models. However, surfaces represented by 3D meshes may contain noise or some unrequired details. Multiresolution representations and filtering techniques are very useful in this case. In this paper, we propose a new and compact representation for the surface of a general 3D mesh using the spherical harmonics. This representation can be useful in many applications such as filtering, progressive transmission and compression of 3D surfaces. First, we present a basic framework for star-shaped objects. Then, we show how to extend this framework to general form meshes using certain segmentation techniques in combination with implicit surface techniques. An interesting feature of our approach is that the computation of the involved spherical harmonics transform is decomposed into the computation of spherical harmonics transforms based on elementary triangles which compose the mesh. This feature shows that the complexity of the computation of the used spherical harmonics transform linearly dependant on the number of triangles of the mesh. We present some experimental results which demonstrate our technique.

Keywords
Spherical harmonics, triangulated mesh, implicit surfaces, filtering, geometric modeling.

1 INTRODUCTION

Polygonal meshes remain the primary representation of 3D models. The recent development tools for scanning and modelling devices allows these meshes to contain finer details. However, some applications need not these high details to be kept, especially when they cannot be distinguished from noise. These details add geometric and topological complexity to the 3D models, which affects model retrieval applications as well as visualization applications. Due to this demand, the multiresolution representation of 3D meshes has been raised in many research areas, especially in the field of digital geometry processing (DGP) [SS01]. Furthermore, multiresolution representation is interesting for compression and progressive transmission purposes.

WSCG 2006 conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

Multiresolution representation and surface filtering have received a renewal of interest during the recent years [Bül02, CDR00, DMSB99, GSS99, ZBS04]. In signal processing community, the fundamental step is based on the construction of the spectrum of frequencies of the surface with respect to a set of basis functions. The low frequency components in the signal correspond to smooth features, and the high frequency components correspond to finer details such as creases, folds and corners. Retaining just lower frequency components suffices to represent and to capture the overall perceptual shape of the model.

Analyzing 3D meshes for retrieval purpose using the spherical harmonics has been developed during the last decade. As in [FMK+03, KFR03, KFR04, SV01], the spherical harmonics transform of spherical functions induced by the mesh can be calculated using a voxelization of the mesh as a preprocessing step. However, representing and filtering the 3D models using the spherical harmonics have become very interesting thanks to the efforts of Zhou et al. [ZBS04]. They calculate a frequency-based representation of 0-genus meshes using a spherical harmonics transform of spherical conformal parametrization of the mesh. The extension to higher genus meshes is performed by

operating a prior surface cutting along some user-specified closed paths to form a surface with the same topology as the sphere. However, cracks may occur along the cutting boundaries after filtering. Moreover, they consider a separate spherical function for each coordinate component x, y and z of the points of the mesh. They filter each function of $x(\theta,\varphi)$, $y(\theta,\varphi)$ and $z(\theta,\varphi)$ independently of the others without considering the dependencies between these three functions on the surface. Moreover, using three independent functions without regarding the correlation between these functions on the surface causes information redundancy.

Spherical harmonics are not the only approach used to represent the surface at several levels of details. There is also the spherical wavelet techniques [EDD+95, JDBP04] which rely on the same framework as Zhou et al.[ZBS04] except that they applied the spherical wavelet transform instead of spherical harmonics transform. Another example is the Laplacian operator [Tau95] which can smooth large meshes quickly, however, due to the huge computation of the eigenvectors decomposition, this method is limited to low pass filtering and cannot be used for general filter design.

Our contribution

In this paper, the spherical harmonics transform of a radial function induced by a mesh is computed in a cumulative manner. That is, the transform is applied independently to each triangle representing the mesh and then the results are summed up.

0-genus object can be represented by a set of three spherical functions [ZBS04] thanks to a previous conformal parametrization, however, in the case of a star-shaped object, a single spherical function is enough and does not require any previous parametrization. Therefore, there no information redundancy as the case of using three independent spherical functions in [ZBS04]. Moreover, it allows to exploit the correlation between the x, y and z coordinates on the surface. In case that the object is not star-shaped, we decompose it into star-shaped parts by a robust segmentation method, and show how a frequency-based description of the whole object can be obtained from the frequency-based description of the parts. For that, we revert to an implicit formulation for each part and blending techniques to extend to the entire object.

This paper is organized as follows. Section 2 recalls a brief mathematical background of the spherical harmonics. Section 3 shows how to represent star-shaped objects using the spherical harmonics transform that is calculated directly on the mesh without voxelization. Section 4 gives a generalization of our method to represent general meshes using the spherical harmonics by the means of the segmentation and the implicit framework techniques. Section 5 shows the use of our surface representation to filter general models using the spherical harmonics. We give some experimental results that demonstrate our approach in section 6. Finally, we conclude in section 7.

2 BACKGROUND

Spherical harmonics $\{Y_l^m(\theta,\varphi) : |m| \leq l \in \mathbb{N}\}$ are special functions defined on the unit sphere $\mathbb{S}^2$ [Bye59, Hob55] as :

$$Y_l^m(\theta,\varphi) = (-1)^m k_{l,m} P_l^m(\cos\theta) e^{im\varphi} \qquad (1)$$

where $\theta \in [0\ \pi]$, $\varphi \in [0\ 2\pi]$, $k_{l,m}$ is a constant, and P_l^m is the associated Legendre polynomial. The spherical harmonics are orthonormal functions such that:

$$\int_0^{2\pi}\int_0^{\pi} Y_l^m(\theta,\varphi)\overline{Y}_{l'}^{m'}(\theta,\varphi)\sin(\theta)d\theta d\varphi = \delta_{l,l'}\delta_{m,m'} \qquad (2)$$

where $\delta_{u,v}$ is the Kronecker delta function defined as the following :

$$\delta_{u,v} = \begin{cases} 1 & if\ u = v; \\ 0 & otherwise. \end{cases} \qquad (3)$$

Therefore, any spherical function $f : \mathbb{S}^2 \longrightarrow \mathbb{R}$ can be expanded as a linear combination of spherical harmonics :

$$f(\theta,\varphi) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} c_{l,m} Y_l^m(\theta,\varphi) \qquad (4)$$

where the coefficients $c_{l,m}$ are uniquely determined by :

$$c_{l,m} = \int_0^{2\pi}\int_0^{\pi} f(\theta,\varphi)\overline{Y}_l^m(\theta,\varphi)\sin(\theta)d\theta d\varphi \qquad (5)$$

Since f is a real valued function, the coefficients $c_{l,m}$ are related to each other by the following relation :

$$c_{l,-m} = (-1)^m \bar{c}_{l,m} \qquad (6)$$

3 SPHERICAL HARMONIC REPRESENTATION OF STAR-SHAPED OBJECTS

Let M denote a triangulated mesh of an object embedded in $\mathbb{R}^3$. M is said to be star-shaped if there exists a point $c \in \mathbb{R}^3$ such that every line segment drawn from c in any direction intersects the surface of M at exactly one point. Considering that c is the center of the spherical coordinate system, the radial function induced by M and c is a well-defined spherical function $f : \mathbb{S}^2 \longrightarrow \mathbb{R}^+$, where $\mathbb{S}^2$ is the unit sphere. More formally, this function is

defined as the following: for each (θ, φ) let p be the intersecting point with M in the direction (θ, φ)

$$f(\theta, \varphi) = d(c, p) \tag{7}$$

where d is the euclidean distance.

In this section, we will show how to represent a star-shaped object using the spherical harmonics. To do this, we propose to represent the object by its radial function $f(\theta, \varphi)$ with respect to a centre c. The spherical harmonics transform (SHT) is applied to this radial function taking into account that the surface is made of triangles.

3.1 Spherical harmonics transform of a star-shaped mesh

Let M denote a star-shaped triangulated mesh with respect to a point c. In the case that f is a binary function, standard algorithms [KFR03, FMK+03] compute a voxelization of M and then use this discrete approximation to find the coefficients of the spherical harmonics transform of f. The discretization introduces uncontrolled errors in the integrations needed to compute the harmonic coefficients.

In [MCA] we have proposed a fast and robust algorithm to calculate the spherical harmonics transform of triangulated meshes indicator function without prior voxelization, extending the decomposition idea proposed in [ZC01]. The calculations are performed independently over the triangles of M and then are summed up to obtain the final transform of M. We can apply the same algorithm to perform the spherical harmonics transform for the radial function defined by equation 7. Assuming that $f_i(\theta, \varphi)$ is the partial radial function defined over the triangle i with respect to a point c, the global radial function $f(\theta, \varphi)$ can be decomposed in terms of $f_i(\theta, \varphi)$ as follows :

$$f(\theta, \varphi) = \sum_{i \in T} f_i(\theta, \varphi) \tag{8}$$

where T is the set of triangles of M. The spherical harmonics transform of f_i is given by :

$$f_i(\theta, \varphi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{l,m}^i Y_l^m(\theta, \varphi) \tag{9}$$

Therefore, the expansion of $f(\theta, \varphi)$ can be rewritten as follows :

$$f(\theta, \varphi) = \sum_{i \in T} \left(\sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{l,m}^i Y_l^m(\theta, \varphi) \right) \tag{10}$$

3.2 Approximating the signal

Theoretically, the expansion of $f(\theta, \varphi)$ is an infinite sum of the spherical harmonics. However, the high order coefficients $c_{l,m}$ obtained by summing up the $c_{l,m}^i$ are corresponding to finer details of the surface, and maybe noise. To filter the surface under a given precision, the summation is limited to a bandwidth bw. We then obtain an approximated surface.

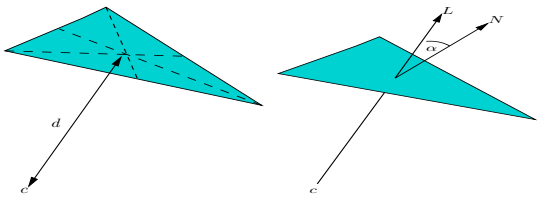
$$\hat{f}(\theta, \varphi) \approx \sum_{i \in T} \left(\sum_{l=0}^{bw} \sum_{m=-l}^{l} c_{l,m}^i Y_l^m(\theta, \varphi) \right) \tag{11}$$

We introduce an error measure ε between the approximated signal $\hat{f}(\theta, \varphi)$ and the original signal $f(\theta, \varphi)$. ε is defined as follows :

$$\varepsilon = \sqrt{\sum_{j \in V} \left[f(\theta_j, \varphi_j) - \hat{f}(\theta_j, \varphi_j) \right]^2} \tag{12}$$

where V denotes the set of points of M. To have a more accurate precision, we can take V as the union of the set of points of M and the set of centroids of the triangles of M. The error ε can be calculated exactly, it depends on the value of bw. The greater the value of bw the smaller the value of ε. Restricting the error measure to the level of a triangle, we can make it as small as desired by increasing bw. The value of bw mainly depends on two factors :

- the distance d between the centroid of the triangle and the point c, Figure 1(a),
- the angle α between the normal of that triangle and the line connecting c and the centroid of the triangle, Figure 1(b).



(a) distance (b) angle

Figure 1: The orientation and the distance of a triangle with respect to the point c.

Figure 2 gives a graphical analysis of the value of bw with respect to the two previous factors, for a fixed quality error ϵ. When the distance d increases, the projecting area of the triangle on the unit sphere decreases. Therefore, the bandwidth bw has to increase so as to compensate the distortion of the triangle due to this decreasing of the projecting area. In a similar manner,

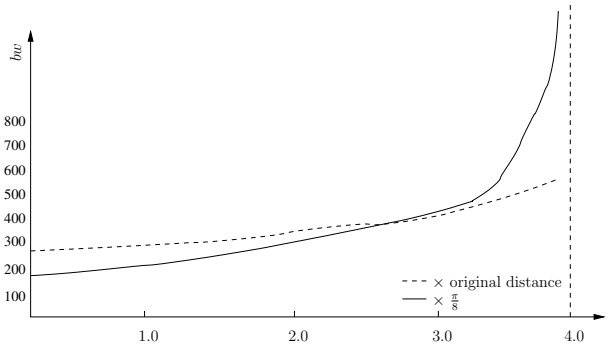
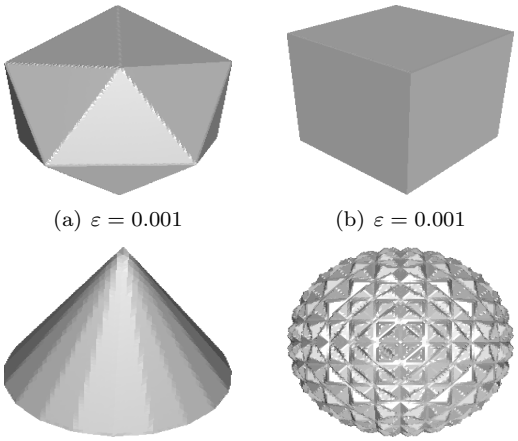


Figure 2: The effect of the distance and the orientation angle of a single triangle with respect to a fixed point c.

the bandwidth bw increases when the angle α increases. The triangle has a maximum distortion when $\alpha = \frac{\pi}{2}$. Figure 3 shows some examples representing some star-shaped models using the spherical harmonics. The bandwidth bw in this case has been fixed to 64 and the center c is the center of mass of the model. On those examples, we observe that $\varepsilon \leq 0.005 * \frac{D}{2}$, where D is the diagonal of the bounding box of the model.



(a) $\varepsilon = 0.001$ (b) $\varepsilon = 0.001$

(c) $\varepsilon = 0.002$ (d) $\varepsilon = 0.005$

Figure 3: Star-shaped objects represented by the spherical harmonics transform of their radial functions, $bw = 64$.

4 FREQUENCY-BASED REPRESENTATION OF GENERAL MODELS

In this section, we extend our framework for representing general 3D models using spherical harmonics. This method consists of three main steps. In the first step, we segment the object into star-shaped parts using a robust segmentation technique [DGG03]. In the second step, we apply the spherical harmonics transform to each part separately as described in section 3. In the last step, we represent each filtered part as an implicit surface, and the whole object is obtained by blending together these implicit representations.

4.1 Segmentation

There are many decomposition techniques used to break complex models into convex or star-shaped sub-models. Lien et al. [LA04] have proposed a concavity measure to partition the models into approximately convex pieces, according to that measure. Dey et al. [DGG03] have decomposed the volume enclosed by a set of points into smaller sub-volumes. Their segmentation is based on topological persistence [ELZ00]. Firstly, they find the critical points of the distance function to the nearest point of the model and they classify them as maxima, minima and saddle points. Then they define what they call stable manifolds based on those maxima. Those stable manifolds are compact regions and decompose the interior volume of the model. Moreover, they are often convex regions.

In this paper, we have used this latter technique. Its advantage is that it also offers a good candidate for the choice of a center (the maximum attached to the region of manifold). However, since further fusions can be performed by their segmentation algorithm between the obtained parts, the choice of one center is less direct. This is the reason why we took the center of mass of the region instead. Nevertheless, this decision could be improved. After the choice of the center for each part, the latter is represented by its radial function $f : \mathbb{S}^2 \rightarrow \mathbb{R}$ with respect to this center.

Note that the initial object may be star-shaped with respect to a point c, but a great number of its triangles may not have a good orientation with respect to c, see Figure 4. So in this case, it is

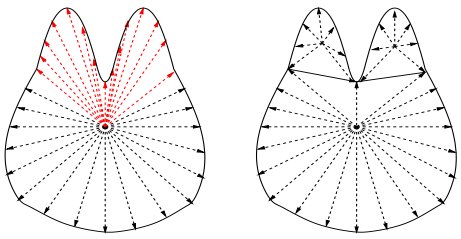


Figure 4: Left: bad orientation of some triangles (red) of a star-shaped object, right: improving the orientation of the triangles by segmenting the object.

recommended to decompose the object into simpler parts. This case can be detected by determining the ratio between the farthest and the nearest points of the object from c. If this ratio is greater than a threshold, then we decompose the object using the technique of Dey et al.[DGG03].

4.2 Conversion into implicit representation

The segmentation of the object results in a finite number of sub-objects, each of which can be represented by a radial function $f : \mathbb{S}^2 \rightarrow \mathbb{R}$ with respect to a center. Consider that $\{M_i : \ i = 1, \ldots, n\}$ is the set of sub-parts and let $\{f_i(\theta, \varphi) : \ i = 1, \ldots, n\}$ denote their radial functions with respect to the set of points $\{c_i : \ i = 1, \ldots, n\}$ respectively. Recall that each $f_i(\theta, \varphi)$ measures the extent of M_i from c_i in the direction (θ, φ).

Consider $\{g_i(r, \theta, \varphi) \ i = 1, \ldots, n\}$ as the set of functions defined as the following : for each point $q \in \mathbb{R}^3$ whose coordinates with respect to c_i are $(r_q^i, \theta_q^i, \varphi_q^i)$

$$\begin{array}{rcl} g_i(r_q^i, \theta_q^i, \varphi_q^i) & = & f_i(\theta_q^i, \varphi_q^i) - d(c_i, q) \\ & = & d(c_i, p) - d(c_i, q) \end{array} \quad (13)$$

where p is the intersection point of M_i with the line $\overrightarrow{c_i q}$, and d is the euclidean distance. Each g_i has the following property :

$$sign(g_i(q)) = \begin{cases} + & if\ q\ is\ inside\ M_i, \\ - & if\ q\ is\ outside\ M_i, \\ 0 & if\ q\ is\ on\ M_i. \end{cases} \quad (14)$$

Therefore, the surface of M_i can be considered as the 0-level of the potential function g_i. The volume of the whole object M is the union of the volumes of these implicit surfaces. The theory of R-functions [PS95, Rva87] provides a useful set of operations on the potential functions. The union operation of two potential functions g_1 and g_2 is defined as follows :

$$g_1 \cup g_2 = \frac{1}{1+a}\left(g_1 + g_2 + \sqrt{g_1^2 + g_2^2 - 2a g_1 g_2}\right) \quad (15)$$

where $a(q) = s(g_1(q), g_2(q))$, s is an arbitrary continuous function satisfying the following condition:

$$-1 < s(t_1, t_2) \leq 1$$

The max function is a special case with $s = 1$. Let M_1 and M_2 be two neighboring parts represented by the radial functions f_1 and f_2 corresponding to the centers c_1 and c_2, respectively. Let g_1 and g_2 denote the corresponding potential functions induced from these radial functions. Therefore, before applying the spherical harmonics transform, we have for any point $q \in \mathbb{R}^3$:

$$g_1(q) = f_1(\theta_q^1, \varphi_q^1) - d(c_1, q) \quad (16)$$

$$g_2(q) = f_2(\theta_q^2, \varphi_q^2) - d(c_2, q) \quad (17)$$

Using equation 15, the potential function g representing the union of the parts represented by g_1 and g_2 is defined as the following:

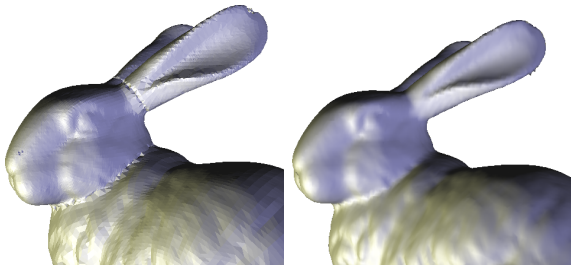
$$g = g_1 \cup g_2 \quad (18)$$

Let $\hat{f}_1$ and $\hat{f}_2$ denote the spherical harmonics transform of f_1 and f_2, respectively. Therefore after restricting the frequencies to a bandwidth bw, we have for any point $q \in \mathbb{R}^3$:

$$\hat{g}_1(q) = \hat{f}_1(\theta_q^1, \varphi_q^1) - d(c_1, q) \quad (19)$$

$$\hat{g}_2(q) = \hat{f}_2(\theta_q^2, \varphi_q^2) - d(c_2, q) \quad (20)$$

The potential function $\hat{g}$ representing the union of the parts represented by $\hat{g}_1$ and $\hat{g}_2$ may have unsmoothness along the boundaries shared by the parts due to restricting the frequencies to a bandwidth bw, as shown in Figure 5(a). To overcome



(a) (b)

Figure 5: Left: unsmoothness along the segmentation boundaries, right: using the blending.

this problem, we apply the set-theoretic blending operator based on R-functions [PS94, PS95]. The corresponding blending operator of the two potential functions $\hat{g}_1$ and $\hat{g}_2$ is defined as follows :

$$\hat{g}_1 \oplus \hat{g}_2 = R(\hat{g}_1, \hat{g}_2) + u \quad (21)$$

where R is the corresponding R-function (the union in our case), and $u(q) = w(g_1(q), g_2(q))$, w is a displacement function that has a maximal absolute value $w(0, 0)$; i.e. at the boundary, and asymptotically approximates a zero value with increasing absolute values of the arguments. The general form of w is as follows [PS94, PS95, Rva87]:

$$w(t_1, t_2) = \frac{1.0}{1 + (t_1/a_1)^2 + (t_2/a_2)^2} \quad (22)$$

where a_1 and a_2 are constants which control the blending operator. We need to choose them optimally with respect to the object.

The error between $g = g_1 \cup g_2$ and $\hat{g}_1 \oplus \hat{g}_2$ is defined as follows:

$$\varepsilon = \sqrt{\sum_{v \in V} (g(v) - (\hat{g}_1 \oplus \hat{g}_2)(v))^2} \quad (23)$$

where V is the set of vertices. By varying the values of the constants a_1 and a_2, we can consider ε as a function of these values. So the objective is to minimize the error function $\varepsilon(a_1, a_2)$. In this paper, we use the genetic algorithms [Gol89] as

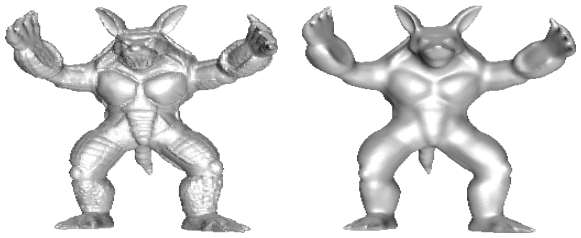
a minimization tool on $\varepsilon(a_1, a_2)$ to choose heuristically the two parameters a_1 and a_2 to remove the unsmoothness along the boundary of the parts. This method gives very good results as we can see in Figure 5(b).

5 APPLICATION : FILTERING

One of the important points of our frequency-based representation is that it allows to filter the surface of n-genus 3D object easily. Surface filtering is useful in smoothing and noise removal [Tau95, GSS99, KG00]. The underlying assumption is that high order frequencies correspond to noises or finer details of the surface. Therefore, removing those frequencies yields a removing of noises or some finer details of the surface. Here, the required filter is applied separately to each triangle of the object and the results are combined as explained previously. Zhou et al. [ZBS04] have presented some interesting frequency filtering functions $h(l, m)$. For example, the ideal low-pass filtering can be performed by setting :

$$h(l,m) = \begin{cases} 0 & if\ \sqrt{l^2+m^2} > K_l \\ 1 & otherwise \end{cases} \qquad (24)$$

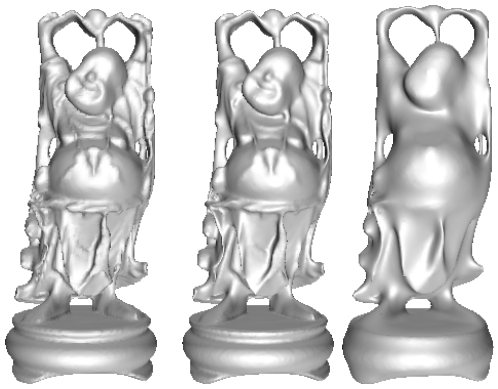
Figure 6 shows smoothing the surface of Armadillo using the previous low-pass filter.



(a) original (b) $K_l = 60$

Figure 6: Smoothing the surface of Armadillo using ideal low-pass filter.

Additionally, surface filtering is also useful for compression and progressive transmission of 3D models. The underlying assumption is that a relatively good approximation may be obtained using only a small number of low-frequency basis functions, see Figure 7. That is, we can send a small number of low-frequency coefficients through the network and progressively send the higher ones to have finer details. Figure 7 shows levels-of-details of Happy Buddha. These levels-of-details can be considered as compressed versions of the model. For example, Figures 7(b) and 7(c) have compression ratios 87% and 99.3% respectively with regards to an initial OFF (Object File format) representation, and without further compression of the obtained sequence of coefficients.



(a) original (b) $K_l = 256$ (c) $K_l = 60$

Figure 7: Several levels-of-details of Happy Buddha using ideal low-pass filter. (a) corresponds to 543,652 points described by doubles and 1,087,716 faces, (b) and (c) correspond to $\frac{(k_l)^2}{2}$ coefficients described by complexes. The compression ratios for (b) and (c) are 87% and 99.3% respectively.

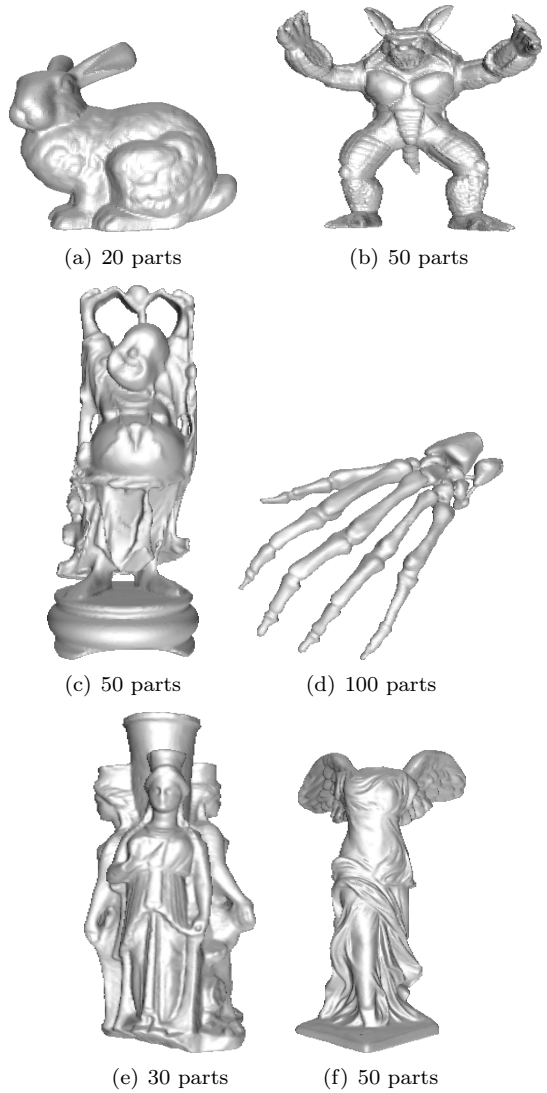
6 EXPERIMENTAL RESULTS

Our method is implemented in C++; we have used a 3GHz Pentium IV PC with 1GB memory for the experiments. The input meshes are considered triangulated. Otherwise, a preprocessing step is required to triangulate the polygons of the mesh. The segmentation step does not take more than 4 minutes for each of the models used in this paper. Generally, the number of parts is in average 50 parts. We have visualized our result by reconstructing the surfaces of the models using the marching cube algorithm proposed by Lewiner et al. [LLVT03]. Table 1 shows a summary of the experimental results.

Model	no. of triangles	SHT time	no. of parts
Bunny	69,451	3min	20
Triple Hecate	180,364	5min	30
Victoire	187,072	6min	50
Buddha	1,087,716	8min	50
Armadillo	345,944	7min	50
Hand	654,666	5min	50

Table 1: Summary of the models used in this paper.

Figure 8 presents some examples of general models, represented using spherical harmonics frequencies. Each model is segmented into subparts, each of which is transformed into a set of frequencies using the spherical harmonics. Taking $\varepsilon \leq 0.01 * \frac{D}{2}$, where D is the diagonal of the bounding box, the corresponding bandwidths are 128, 256, 256, 128, 256 and 256 for the Bunny, Armadillo, Happy Buddha, Hand, Triple Hecate and Victoire respectively.



(a) 20 parts (b) 50 parts

(c) 50 parts (d) 100 parts

(e) 30 parts (f) 50 parts

Figure 8: Some models represented using the spherical harmonics. The bandwidth bw has the following values (a) 128 (b) 256 (c) 256 (d) 128 (e) 256 and (f) 256.

7 CONCLUSION AND FUTURE WORK

This paper presents a new technique to represent general 3D models using the spherical harmonics. This representation allows us to filter the surfaces of these models, although they are not topologically equivalent to the sphere, and to describe them compactly. The spherical harmonics transform is computed using a fast, combinatorial and robust algorithm [MCA]. The implicit framework techniques guarantee the avoidance of unsmoothnesses on the surfaces.

Concerning the segmentation technique, the technique proposed in [DGG03] is robust and yields a good segmentation of the objects. However, we need as well to have an optimal segmentation technique that produces approximately convex (or star-shaped) sub-parts while keeping the number of parts as small as possible.

On the other hand, we have chosen the center of each part as the center of the mass of that part. Since the radial function of each part is dependent on the choice of the center, the final representation is affected by this choice. We work to improve this choice of the center.

ACKNOWLEDGEMENT

We acknowledge Dey et al. for providing us with their code of the segmentation algorithm. We also acknowledge the support of ART3D project funded by the french ministry of research. The model of Triple Hecate is one of the statues disponible at the Louvre Museum [Lou]. Thanks to the Stanford graphics laboratory website and the large geometric models archive of Georgia Institute of Technology for putting many models at disposal.

REFERENCES

[Bye59] W. E. Byerly. *Spherical Harmonics*, chapter 6, pages 195–218. New York: Dover, 1959. An Elementary Treatise on Fourier's Series, and Spherical, Cylindrical, and Ellipsoidal Harmonics, with Applications to Problems in Mathematical Physics.

[Bül02] T. Bülow. Spherical diffusion for 3d surface smoothing. In *3DPVT'02: the first International Symposium on 3D Data Processing Visualization and Transmission*, page 449, Padova, Italy, 2002.

[CDR00] U. Clarenz, U. Diewald, and M. Rumpf. Nonlinear anisotropic diffusion in surface processing. In T. Ertl, B. Hamann, , and A. Varshney, editors, *Proceedings of IEEE Visualization 2000*, pages 397–405, 2000.

[DGG03] T. K. Dey, J. Giesen, and S. Goswami. Shape segmentation and matching with flow discretization. In F. Dehne, J.-R. Sack, and M. Smid, editors, *WADS '03: Proceedings of the* 8^{th} *International Workshop on Algorithms and Data Structures*, number 2748 in LNCS, pages 25–36, Carleton Univ., Ottawa, Canada, July-August 2003. Springer Verlag.

[DMSB99] M. Desbrun, M. Meyer, P. Schroder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 173–182, New York, NY, USA, 1995. ACM Press.

[ELZ00] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 454, Washington, DC, USA, 2000. IEEE Computer Society.

[FMK+03] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3d models. *ACM Transactions on Graphics*, 22(1):83–105, January 2003.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co., 1989.

[GSS99] I. Guskov, W. Sweldens, and P. Schroder. Multiresolution signal processing for meshes. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 325–334, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Hob55] E. W. Hobson. *The Theory of Spherical and Ellipsoidal Harmonics*. New York: Chelsea, 1955.

[JDBP04] J. Jin, M. Dai, H. Bao, and Q. Peng. Watermarking on 3d mesh based on spherical wavelet transform. *Journal of Zhejiang University Science*, 5(3):251–258, 2004.

[KFR03] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164. Eurographics Association, 2003.

[KFR04] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Symmetry descriptors and 3d shape matching. In *SGP '04: Symposium on Geometry Processing*, pages 116–125, July 2004.

[KG00] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[LA04] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 457–458, New York, NY, USA, 2004. ACM Press.

[LLVT03] T. Lewiner, H. Lopes, A. Wilson Vieira, and G. Tavares. Efficient implementation of marching cubes cases with topological guarantees. *JGT (Journal of graphics tools)*, 8(2):1–15, 2003.

[Lou] The Louvre Museum. http://www.louvre.fr/.

[MCA] M. Mousa, R. Chaine, and S. Akkouche. Direct spherical harmonic transform of a triangulated mesh. *JGT: Journal of Graphics Tools*. to appear.

[PS94] A. Pasko and V. Savchenko. Blending operations for the functionally based constructive geometry. In *CSG 94: Set-theoretic solid modelling–Techniques and Applications*, pages 151–161, Winchester, UK, April 1994.

[PS95] A. Pasko and V. Savchenko. Constructing functionally defined surfaces. In B. Wyvill and M. P. Gascuel, editors, *Implicit Surfaces '95: The first international workshop on implicit surfaces*, pages 97–106, Grenoble, France, April 18–19 1995.

[Rva87] V. L. Rvachev. *Theory of R-functions and some applications*. Naukova Dumka, Kiev, 1987. (in Russian).

[SS01] W. Sweldens and P. Schröder. Digital geometric signal processing, course notes 50. In *SIGGRAPH 2001 Conference Proceedings*, 2001.

[SV01] D. Saupe and D. V. Vranic. 3d model retrieval with spherical harmonics and moments. In *Proceedings of the 23rd DAGM-Symposium on Pattern Recognition*, pages 392–397. Springer-Verlag, September 2001.

[Tau95] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, NY, USA, 1995. ACM Press.

[ZBS04] K. Zhou, H. Bao, and J. Shi. 3d surface filtering using spherical harmonics. *Computer-Aided Design*, 36(4):363–375, 2004.

[ZC01] C. Zhang and T. Chen. Efficient feature extraction for 2d/3d objects in mesh representation. In *ICIP '01: Proceedings of the International Conference on Image Processing*, pages 935–938, October 2001.

Learning to Synthesize Arm Motion to Music By Example

Sageev Oore
Saint Mary's University, Halifax, Canada
sageev@cs.smu.ca

Yasushi Akiyama
Saint Mary's University, Halifax, Canada
y_akiyama@cs.smu.ca

ABSTRACT

We present a system that generates arm dancing motion to new music tracks, based on sample motion captured data of dancing to other pieces of music. Rather than adapting existing motion, as most music-animation systems do, our system is novel in that it analyzes both the supplied motion and music data for certain characteristics (eg. melodic contour, loudness, etc), and learns relationships between the two. When new music is provided, the characteristics are analyzed as before, and used to predict characteristics of the motion. A generative process then creates motion curves according to these constraints. A demonstration is presented showing the results on both slow and fast tunes, including an automatically-generated trio of backup dancers for a jazz standard. The system could be extended naturally to other movements beyond arm dancing.

Keywords: animation, learning, motion/music synchronization.

1 INTRODUCTION

There are a myriad ways to create character motion to music. Certain characters may have a particular styles of dancing, based both on their ways of moving, and also on how they perceive the music, or based on the directions of a choreographer. How does the nature of some of these dance motions— especially in the "background" contexts that one might want to automate— relate to musical characteristics such as loudness, pitch, and note density? There is no constant relationship, of course, but for a given context or character, there may likely be a particular connection or association. Our paper presents an initial approach towards answering this question by analyzing the characteristics of music and motion data, and then by providing a prototype tool for animators to specify some of these relationships implicitly by example.

Various work [ES03, Lyt90, LL05, CBBR02, KPS03, WLXS02, GBBM96, GM95, PN03] has been focused on the problem of adapting existing motions to music. In many of these systems, the primary mappings between the features of music and motion are specified explicitly by the user. In others, motions are synthesized from an existing database. This research proposes a system for learning the characteristics of desired motions associated with musical phrases by user-provided examples for some musical phrases, in order to produce an animated figure that responds not only to the musical phrases that are used to train it, but will also be able to handle new musical ideas. Such a tool would be intended for use by animators to automatically choreograph characteristics of dance for a musical performance, so the final motions can be created automatically. For example, ultimately the dance style in a certain nightclub or a set of backup singers/dancers for a musical act, or recurring dancers in a video game, could be done in this way.

Since the motion-music relationships are inferred from the data, it follows that different data sets will lead to different dance styles. For example, if all of the supplied training data were to depend exclusively on the pitch, and not be affected in any way by changes in loudness (e.g. the 'dynamics of the phrasing', in musical terms), then this should be visible in the resulting animation. The system is not intended to learn a single definitive relationship between motion and music based on a large comprehensive motion database, but rather to allow certain relationships– as needed for a particular set of characters or situations– to be shown by example. In Section 9 we demonstrate this with automatically generated arm motions for a set of backup dancers for a jazz standard.

2 RELATED WORK

The issue of synchronizing animation to music has been addressed by a number of researchers. Kim et al. [KPS03] and Alankus et al. [ABB04] created systems using motion graphs [KGP02] and beat analysis to synthesize a motion from existing sample motions synchronized to background music. The systems by Cardle et al. [CBBR02] and Lee and Lee [LL05] synchronize motion curves by locally modifying motions using perceptual cues obtained from the music. Lytle's [Lyt90] system creates animation of musical instruments from orchestrated MIDI music. Goto and Muraoka [GM95] have multiple musicians, each assigned to control specific features of a single animated character. Penasse

WSCG 2006 conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

and Nakamura [PN03] used the analog sound signal and adjusted key frames so that they fall on the beats of music. In these systems, using explicit mappings of features between music and animation seems like a standard approach. ElKoura and Singh [ES03] use input music data in an augmented tablature notation for guitar. Their system solves the problem of multiple concurrent reaching tasks by finding the likely hand configurations in the realistic hand configurations obtained from example motions, and minimizing a cost function for hand motions. Wang et al [WLXS02] predict the joint angle trajectories to create conducting motions by using kernel-based Hidden Markov Models. Learning from the music with the time signature that has the same or similar musical accents and relying on beat information provided in the MIDI file may limit its flexibility of creating motions responding to music that is played more freely.

3 SYSTEM OVERVIEW AND OUTLINE

The goal of our system is to accept examples of synchronized motion and music, in order to learn a relationship between the two, and subsequently use this information to generate animation for new music input. For demonstrative purposes, the present system focuses on arm motions recorded to melodic lines in MIDI format; scalability is discussed in Section 10.

We now give an outline of this paper, along with an overview of the two phases, in which our system operates.

1) Training Phase: Fig 1

A set of examples is provided consisting of hand-motion recorded in synchrony with music in MIDI format (described in Section 4). Based on our motion model, the motion data is analyzed for certain characteristics: distance-from-body, amplitude, centres-of-motion (described in Section 5). Likewise, the music data is analyzed for characteristics such as loudness and note density (Section 6). A relationship between the input and output characteristics is learned so as to be able to predict a distribution over likely output motion characteristics given music input characteristics (Section 7).

2) Generative Phase: Fig 2

New music input is provided. The system analyzes it as before, and uses the learned model to stochastically generate a set of motion characteristics. A generative process is applied to create motion curves for the final animation (Section 8).

4 DATA COLLECTION

Music Data .

Music is input in a standard MIDI file format. Each note event contains information pertaining to pitch, loudness, time, and a NOTE ON and NOTE OFF signal, indicating whether the note is starting or stopping. The MIDI format we use is Type 1, which allows a file to have multiple tracks. This is convenient as it lets us focus directly on processing individual lines (eg. melody, bass) and avoid the task of separating a given score into parts [SNK01].

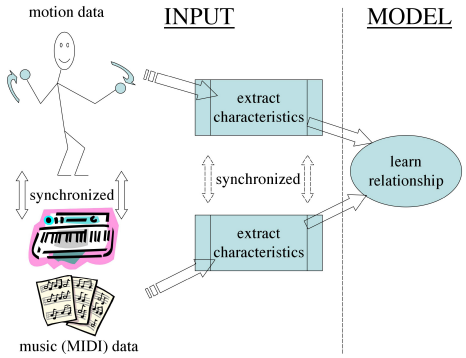


Figure 1: Training Phase: Synchronized examples of motion and music are provided to the system.

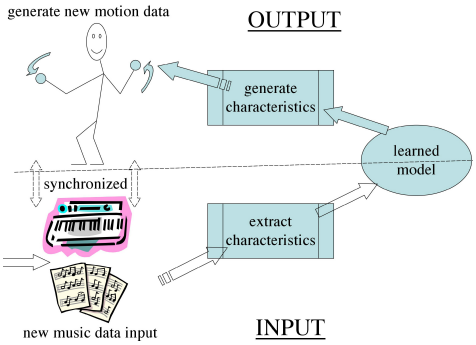


Figure 2: Generative Phase: New music is now provided, and using the learned model, motion is now the output rather than input (as indicated by the reversed arrows).

Motion Capture Two 3D motion capture devices are used to collect the sample motion data. The dancer familiarizes herself with the music prior to the motion recording session. We acquire 3 degrees of freedom (DOF) from each sensor, measuring hand position. As the dancer is aware that the motion will be used for the arm motion of a character who is standing in one spot during recording, the dancer also remains in one spot to ensure that the motion examples are representative.

5 DANCE MOTION ANALYSIS

Our motion model is developed based on a few key properties observed in the captured sequences. In the following discussion, let $B_1, B_2, \ldots, B_N$ represent N different recorded motions, each of which was danced in sync to the same music track M.

5.1 Spatial Characteristics

An important visible feature of the arm motion is *extent*: the distance of the hands from the body [Lab88, NF03].

We first compute $D_k(t)$, the Euclidian distance of the hands from the root node at time t, for motion B_k, where the root can simply be approximated by a point near the middle the torso for this purpose. We then estimate the *global motion centre* by computing the mean position of the entire hand motion. Finally, we compute $dist_k(t)$, the Euclidian distance of the hands from the global motion centre. This is shown in Figure 3. Figure 4 shows $dist_k(t)$ of four sample captured dance sequences to the same piece of music M.

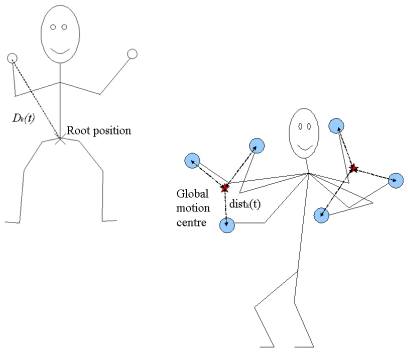


Figure 3: Root position, $D_k(t)$, the global motion centre, and $dist_k(t)$

The motions tend to be comprised of oscillating segments, corresponding to back-and-forth (or side-to-side etc) movement of the hands. We model such oscillations by estimating, for each time frame, an approximate centre of extent about which the current oscillation is taking place, as well as its approximate amplitude. To do this we define a window $V(t_i)$ around time t_i, corresponding to a set of $S+1$ frames of recorded dance motion from $(t_i - S/2)$ to $(t_i + S/2)$. This is shown in Figure 5.

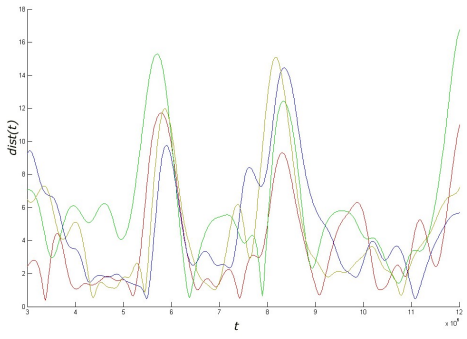


Figure 4: This illustrates 4 different sets of $dist(t)$ for a single music track. Each colour corresponds to one recorded motion. While each curve is different, they share certain characteristics, and those characteristics are what we would like to learn. E.g. the two largest peaks happen around the same time in all the motion samples, and the ranges of the oscillation amplitudes are very similar within a certain period of time. Also, they share similar oscillatory nature.

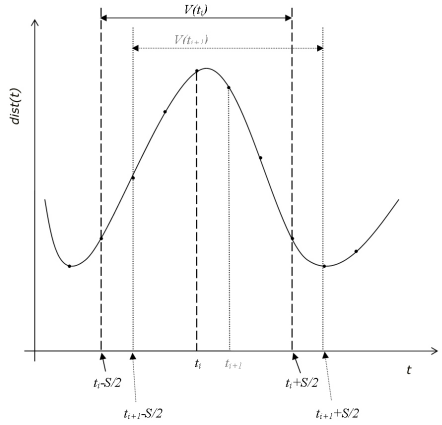


Figure 5: For each frame, $m_k(t_i)$ and $v_k(t_i)$ are computed in the moving window $V(t_i)$.

For each window $V(t_i)$ we compute the mean hand distance (or "center of oscillation"):

$$m_k(t_i) = \frac{1}{S+1} \sum_{t_i \in V(t_i)} (dist_k(t_i)) \qquad (1)$$

Similary, we compute the variance for window $V(t_i)$:

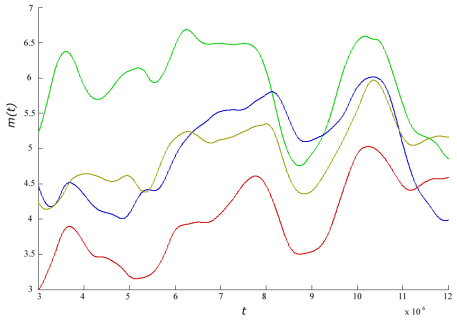


Figure 6: The mean hand distance $m_k(t)$. The same colors correspond to the same motions in the previous and next figures.

$$v_k(t_i) = \frac{1}{S+1} \sum_{t_i \in V(t_i)} (dist_k(i) - m_k(t_i))^2 \qquad (2)$$

The square-root of this value, $\sqrt{v_k(t_i)}$, relates to the amplitude of the oscillations, as it tells us how far the hand moves from the local centre of oscillation. Figures 6 and 7 show $m_k(t)$ and $v_k(t)$, respectively, for clips from a set of motions all corresponding to one piece of music. Looking at these graphs, we see that, for a given piece of music, there are sections where there is considerable variance in the m_k, and other sections

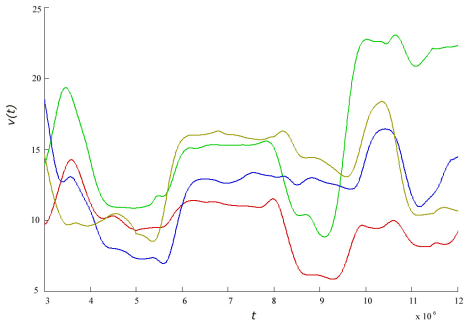


Figure 7: The variance $v_k(t)$

wherein the m_k are relatively tight. Similarly, looking at the estimated amplitudes over the same motion set, more coherence is evident in some areas than in others. This suggests that, if we are to predict $m(t)$ and $v(t)$ for some new piece of music, and if we want to be able to create multiple animations sharing "characteristics", it is not enough to simply predict a single value for $m(t)$, but rather a *distribution* which we will specify by a mean $\mu_m(t)$ and variance $\sigma_m(t)$, from which we will later sample values (Section 8). Similarly we will predict $\mu_v(t)$ and $\sigma_v(t)$. In order to learn to predict these values for a new piece of music, we first estimate them for the given data:

$$\mu_m(t_i) = \frac{1}{N}\sum_{k=1}^{N}(m_k(t_i)) \tag{3}$$

$$\sigma_m(t_i) = \frac{1}{N-1}\sum_{k=1}^{N}(m_k(t_i) - \mu_m(t_i))^2 \tag{4}$$

and

$$\mu_v(t_i) = \frac{1}{N}\sum_{k=1}^{N}(v_k(t_i)) \tag{5}$$

$$\sigma_v(t_i) = \frac{1}{N-1}\sum_{k=1}^{N}(v_k(t_i) - \mu_v(t_i))^2. \tag{6}$$

In Section 6 we will describe some of the music characteristics that may drive $m(t)$ and $v(t)$, but first we consider some spatial characteristics of the motion.

5.2 Temporal Characteristics

While the frequency is of course not constant, there is likely a relationship between the motion and the rhythm of the music, and once again, we will be trying to capture certain characteristics of this relationship. One type of visually salient event is the *peak* of an oscillation— a hand stopping and going in another direction. This event corresponds to zero-crossings of derivative, and we would like to model these *peak points* both in time and in space. The amplitude and oscillation centre predictors $(\mu_v, \sigma_v, \mu_m, \sigma_m)$ will give us, based on the music characteristics, a distribution over where the peak point should be in space; another predictor will be used to stochastically choose the frame at which that peak point should occur so that it is in synchrony with musical characteristics. To provide training data for the learning algorithm, we create, for each motion B_k, a binary variable

$$peak_k(t) = \begin{cases} 1 & \text{if a local extremum occurs in } t \pm \Delta t, \\ 0 & \text{otherwise} \end{cases}$$

where Δt is a small constant $\approx 0.1sec$ to allow for slight temporal discrepancies between the data and the music. Combining $peak_k(t)$ for each B_k, where $k = 1, \ldots, N$, we obtain an estimate, for a given piece of music, of the independent probability of a peak event occuring at each time slice. We repeat this for each of the given music tracks and corresponding sets of animations. This gives us a set of music tracks, each with a corresponding function $peak(t)$ marking possible moments at which such events may occur.

6 MUSIC ANALYSIS

Before we can apply a machine learning tool to predict the motion characteristics described above, we need to consider the 'input' variables to the system. That is, we now extract certain characteristics of a given musical track that may influence the resulting animation. The three key concepts in which we are interested are (1) melodic contour, (2) loudness and (3) note density. To realize this, we need to compute various quantities as described below.

6.1 Dealing with Multiple Scales

Suppose that the current note is somewhat higher (or louder) than the previous note. What is the significance of this, with the view of choreographing a dance to the music? This would depend not only on the previous note, but also on where this note fits within the larger context of the pitch (or loudness) contour. Thus, we consider musical progression at several time-scales, by computing characteristics for three distinct time-windows, $W_1 = [w_0, w_1], W_2 = [w_1, w_2], W_3 = [w_2, w_3]$, and refer to the collection of these subwindows as $W = [w_0, w_3]$, with $w_j < w_{j+1}$ and w_3 being the current frame. This is illustrated in Fig 8.

Note that the windows used here for the music context are different from the windows used for the motion data analysis, in that the music windows are to capture the music contour that leads up to the current note event, while the motion windows are used to capture the behaviour *around* the current event. This is why the music window ends on the current event, but the motion window is centred on the current event.

6.2 Melodic Contour Information

Melodic contour refers to the rise and fall of the pitches. If a passage is played slowly and staccato, then most of

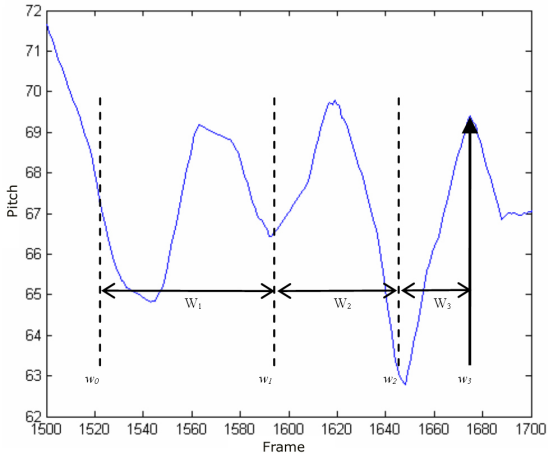


Figure 8: The windows W_1, W_2 and W_3 for multiple scales.

the time there is no "active" note, yet a contour is still implied, and animated motion may correspond to this contour. This can be solved by simply keeping track of the previous note. However, suppose a note is embellished with a trill, tremolo or turn (e.g. notes alternating quickly back and forth). The contour could be considered static in this case as opposed to varying along with the trill. Furthermore, in a fast passage, notes may locally be going up and down, but the overall shape might be a gradually rising progression, so that the musical contour is rising. This motivates a weighted moving-average filter, which also allows the contour value to be expressed as a function of frame number. That is, a contour value is expressed even when there is no note being played. The resolution of the interpolation is determined according to the sampling rate of the motion capture data, so that each motion frame has a corresponding note event. We can then compute the following characteristics:

1. *Pitch samples*. Once the pitches have been processed as described above, processed pitch values $p(t_i)$ can be sampled as needed from W. This input parameter describes the general contour of the music in window W.

2. *Mean and variance of pitches*. Statistics such as mean μ_p and variance σ_p of the pitch contour $p(t)$ can be calculated for each of the windows W_j. μ_p can be considered as a local phrase centre (i.e., around which register the notes occur) while σ_p gives us the idea of how quickly the notes go up (or down) in the given window.

3. *Pitch-peaks*. Just as peaks are salient features of motion, they are salient features of melodic contour, and therefore a pitch-peak variable is used to flag whether the current event is (close to) a local maximum (1), or a local minimum (-1), and 0 is assigned otherwise.

4. *Duration to next/previous closest peaks*. A peak may be perceived to be more significant if it is more isolated from any other nearby peaks. By including these parameters, one can perceive whether or not the peak is isolated from others or just one of many peaks happening in a short period of time.

5. *Phrase endings and duration of phrase*. Although algorithms to detect phrase endings are widely available, a very simple procedure of finding two consecutive notes with the inter-onset time larger than a threshold was used in this study. This method is inspired by Pachet's *Continuator* [Pac02]. The threshold is typically based on the average inter-onset time of all the notes.

6.3 Loudness Characteristics

Loudness information is processed differently from the pitch information. Firstly, we assume that when a note is played, it decays continuously while the key is pressed, and then as soon as the key is released, the note stops. This phenomenon of decaying sound is observed in many of acoustic instruments such as piano and guitar. For non-decaying instruments, we have found it to be sufficient to simply keep loudness constant while the key is being pressed.

For each time frame, we compute a sum of the loudness of all active notes. The loudness $l_j(t)$ of a jth note at time t_i is given by:

$$l_j(t_i) = l_j(t_{j0})\kappa^{(t_i - t_{j0})} \tag{7}$$

where $l_j(t_{j0})$ and t_{j0} are the initial loudness and NOTE ON time of the jth note, respectively, and κ the *decay* constant ($\kappa < 1$). This *cumulative* loudness is relevant because it captures articulation of notes as well as events that are important for realization of the loudness. For example, in the events such as trills and tremolos, the overall loudness is kept near constant as if a single note ringers without decaying even though there are many notes played one after another. In another case when a group of notes are played simultaneously (e.g. chords), the music is usually louder than one-line melodies. Figure 9 shows the result of the cumulative loudness.

Once we have obtained the cumulative loudness of the music, we then extract the information in a manner similar to the process of pitch information. (i.e. loudness samples, mean and variance, peaks, and duration to neighbouring peaks.)

6.4 Density

At each time t_i, the *density* $\rho(t_i)$ of notes is computed for each of the three windows leading up to that time. Let the number of notes in a window $W_k(t_i)$ and the time duration of $W_k(t_i)$ be, $N_k(t_i)$ and $\Delta T_k(t_i)$, respectively, $\rho_k(t_i)$ is computed as:

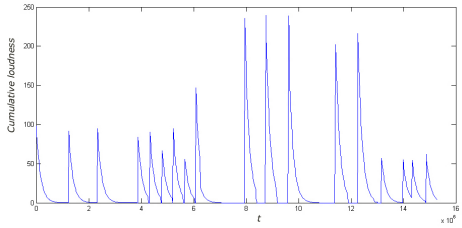


Figure 9: Cumulative loudness processed from MIDI data

$$\rho_k(t_i) = \frac{N_k(t_i)}{\Delta T_k(t_i)} \tag{8}$$

This information describes how many notes are played in a given window, hence, it gives us a general idea of how fast the music is played. Because of the filtering of the pitch information and the cumulation of the loudness, the information of the note density is somewhat lost without explicitly providing it.

7 LEARNING CHARACTERISTIC RELATIONSHIPS

Once the various characteristics have been selected and extracted, a standard neural network model, with tanh functions in the hidden layer, is used to learn relationships based on the provided example data. Other nonlinear models could be also be used.

Our system predicts two types of characteristics of the output motion: temporal and spatial characteristics. The input data is based on a series of music tracks, each with a set of corresponding recorded motions. The input data is processed as described above, so that each time slice of the music contributes one training example. For each frame at time t_i, let $u(t_i)$ represent the set of musical characteristics computed as described in Section 6. One model is trained to predict the probabilities $peak(t_i)$ as described in Section 5 given input $u(t_i)$. The other model learns to estimate the spatial characteristics of the distribution, also described in Section 5.

The temporal model has 22 input parameters (Table 1) and 1 output parameter (Table 3), which is the probability $peak(t)$ indicating the likelihood of the frame at time t being a peak in the motion output. The spatial model has 41 input parameters (Table 2) and 4 output parameters, $\mu_m(t)$, $\sigma_m(t)$, $\mu_v(t)$, and $\sigma_v(t)$ (Table 3), which are used to form distribution functions. We next explain how these output parameters are used to generate motion curves.

8 GENERATING MOTION CURVES

The generative procedure begins by analyzing a new musical input track, and using the $peak(t)$ estimator (predicted by a sigmoidal unit, which can be interpreted as a binary probability value) to stochastically select certain times as moments where motion peaks can occur. The system then goes through all the candidate peak points to make sure that the consecutive peaks are not too close in time. Whether or not they are too close is determined based on the human physical ability. The maximum values of velocity and acceleration are obtained from the example motions. If candidate motion peaks are detected to create movements with higher values than these minimum values, one of them is excluded from the list of peaks. A common previous approach to create animation to music is to assume the beat information is embedded in the MIDI file [KPS03, WLXS02], which also requires an unwavering tempo throughout the piece, or by using beat extraction algorithms such as [Meu02]. However, restricting dance motions synchronizing only to beat is merely one aspect of choreography, and the movements seen in dance are not necessarily accented only by the beat factor. Furthermore, consider music played *rubato* (freedom of tempo) as often found in classical and jazz music, or music in tempo but that has tuplets (e.g. triplets, quintuplets). In these cases, synchronizing the motion with the beat is only one of the possible options; having motion that can be synchronized with articulation of music itself is equally important.

Once the peaks have been selected, then for each peak point at time t_i, the system computes an exact location of the hand as follows. Let $\mu_m(t_i), \sigma_m(t_i), \mu_v(i)$, and

#	Parameter
1-10	Time distances to the last 10 pitch peaks
11	Time distance to the next pitch peak
12 - 21	Time distances to the last 10 loudness peaks
22	Time distance to the next loudness peak

Table 1: Input parameters for the temporal predictor

#	Parameter
1 - 20	Pitch/velocity samples
21 - 23	Note density in each window
24 - 29	Mean/variance of pitch in each window
30 - 35	Mean/variance of loudness in each window
36 - 37	Flags to indicate peaks in pitch/loudness
38 - 39	Time duration of peaks
40	Flags to indicate phrase endings
41	Time duration of phrases

Table 2: Input parameters for the spatial predictor

#	Parameter
1	$\mu_m(t)$
2	$\sigma_m(t)$
3	$\mu_v(t)$
4	$\sigma_v(t)$
5	$peak(t)$

Table 3: Output parameters

$\sigma_v(i)$ be predicted by the model according to the music characteristics $u(t_i)$ at that time. A local motion centre and amplitude for $dist(t_i)$ are chosen by sampling a gaussian distribution with these parameters. Note that the $dist(t)$ function specifies distance of the arm, but does not indicate the direction in which the distance should be chosen. While this is an additional characteristic that could be predicted by the system (discussed in Section 10), we currently randomly set several primary axes of motion $\theta_1, \theta_2, \ldots, \theta_r$ for each character's hands as *mean* directions, and then for any given peak at t_i, we randomly choose one of them, k and sample from a normal distribution centered at θ_k. Together, the axis and distance specifies a point relative to the character.

A quartic polynomial is used to interpolate between the generated peak points. First derivatives are constrained to be 0 at the peaks, and second-derivative continuity is enforced at the join points.

The orientation angles at the characters' links are computed using an IK algorithm.

9 RESULTS

The model was trained on a data set consisting of several pieces of music, each one a few minutes long, together with multiple corresponding samples of recorded hand motion for each piece. Specifically, the average training music sample had 3000-4000 frames, of which we sampled every 10 frames, and used 8 recorded motions per piece, giving roughly 2400 data samples per piece. New pieces of music were then given to the system, which were automatically analyzed, and distribution parameters were predicted for the corresponding motions. Sampling from these parameters and interpolating, as described in above, allowed the creation of multiple motions, sharing certain characteristics, while still differing in the details due to the stochastic nature of the generative process (see the accompanying videos, with sample frames shown in Figure 10). The musical pieces can range from being highly rhythmic, to having rhythmic patterns that shift from slow and rubato to fast, demonstrating that the characters respond appropriately to these complex changes. Testing the system with various pieces shows that the motion can relate to the music even when there is no steady tempo, while at other times it can correspond to certain clear accents in the music. Note that, for each test, the motions of only one dancer were used, so the results corresponded to the patterns specific to that set. This is, indeed, the goal of the system: to learn a mapping based on the motions of a particular choreographer; the system is not intended to learn a general "all-purpose" mapping.

10 CONCLUSION AND FUTURE WORK

We have demonstrated a novel approach to creating animated motion to new musical scores for virtual actors. We have proposed a set of key features of music and motions, and shown how to extract these features. Rather than adapting existing recorded dance motion, we use recorded motions synchronized to music soundtracks, and automatically analyze salient *characteristics* of both the motion and the music. A non-linear parametric model is then used to learn the relationship between these characteristics by example, rather than by requiring an animator to specify mappings explicitly. Once the training data has been specified, then the subsequent musical analysis and stochastic generative process are automated procedures, and thus can be applied in any context where an autonomous animated agent is acting.

Finally, while our present work focuses on animating motion of hand and arm, the same ideas could be extended to include a range of body motion, such as leg and torso motion. In this case, unlike the somewhat free movements of hands, additional constraints would need to be incorporated (e.g. ground contact) in order to maintain physically plausible postures. We expect that further extensions of this sort will contribute significantly to the naturalness of the final synthesized motion.

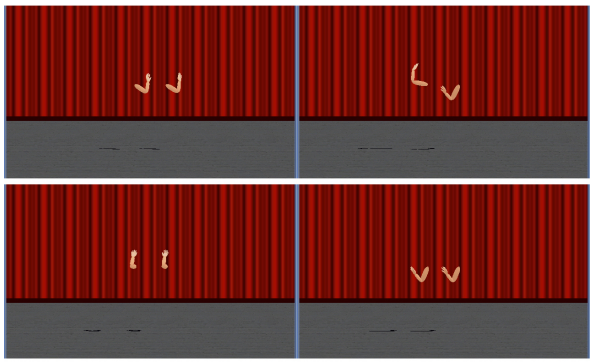


Figure 10: Screenshots of hand motions created stochastically for a new piece of music (see accompanying video).

ACKNOWLEDGEMENTS

We would like to thank Dirk Arnold and the anonymous reviewers.

REFERENCES

[ABB04] Gazihan Alankus, A. Alphan Bayazit, and O. Burchan Bayazit. Automated motion synthesis for virtual choreography. Technical Report WUCSE-2004-66, Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO, 2004.

[CBBR02] Marc Cardle, Loic Barthe, Stephen Brooks, and Peter Robinson. Music-driven motion editing: Local motion transformations guided by music analysis. In *Eurographics UK Conference (EGUK)*, pages 38–44, Leicester, UK, June 2002.

[ES03] George ElKoura and Karan Singh. Handrix: Animating the human hand. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, San Diego, CA, 2003.

[GBBM96] Christian Greuel, Mark T. Bolas, Niko Bolas, and Ian E. McDowall. Sculpting 3d worlds with music; advanced texuring techniques. In *IST/SPIE Symposium on Electronic Imaging: Science & Technology, The enginerring Reality of Virtual Reality III*, 1996.

[GM95] Masataka Goto and Youichi Muraoka. Interactive performance of a music-danced cg dancer. In *Workshop on Interactive Systems and Software (WISS) '95*, 1995.

[KGP02] Lucas Kovar, Michael Gleicher, and Frdric Pighin. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482, New York, NY, USA, 2002. ACM Press.

[KPS03] Tae-Hoon Kim, Sang Ill Park, and Sung Yong Shin. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics*, 22(3):392–401, 2003.

[Lab88] Rudolf Laban. *The Mastery of Movement*. Northcote House, London, 4 edition, 1988. Revised by Lisa Ullman.

[LL05] Hyun-Chul Lee and In-Kwon Lee. Automatic synchronization of background music and motion in computer animation. In *EUROGRAPHICS 2005*, volume 24, September 2005.

[Lyt90] Wayne T. Lytle. Driving computer graphics animation from a musical score. In *Scientific Excellence in Supercomputing, The IBM 1990 Contest Prize Papers*, volume 2, page 644, Ithaca, NY, 1990.

[Meu02] Benoit Meudic. A causal algorithm for beat-tracking. In *2nd Conference on Understanding and Creating Music*, Caserta, Italy, November 2002.

[NF03] Michael Neff and Eugene Fiume. Aesthetic edits for character animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[Pac02] François Pachet. Interacting with a musical learning system: The continuator. In *ICMAI '02:Proceedings of the Second International Conference on Music and Artificial Intelligence*, pages 119–132, London, UK, 2002. Springer-Verlag.

[PN03] Vincent Penasse and Yoshihiko Nakamura. Dance motion synthesis with music synchronization. In *Robotics Society of Japan 2003*, volume 21, 2003.

[SNK01] H.-H. Shih, S. S. Narayanan, and C.-C. J. Kuo. Automatic main melody extraction from midi files with a modified lempel-ziv algorithm. In *International Symposium on Intelligent Multimedia, Video Speech Processing*, 2001.

[WLXS02] Tianshu Wang, Yan Li, Ying-Qing Xu, and Heung-Yeung Shum. Learning kernel-based hmms for dynamic sequence synthesis. In *10th Pacific Conference on Computer Graphics and Applications (PG'02)*, 2002.

Quadrant Motif Approach for Image Retrieval

Tsong-Wuu Lin and Chung-Shen Hung
Department of Computer and Information Science, Soochow University
56 Sec. 1 Kui-Yang St.
Taipei 100, Taiwan, R.O.C.
twlin@cis.scu.edu.tw, ms9315@cis.scu.edu.tw

ABSTRACT

In this paper, we propose an image retrieval approach based on *Quadrant Motif Scan* (QMS). Motif scans from segmented blocks inside an image are the primary notion to extract image features. We exploit recursive quadrant segmentation in images and stratify hierarchical regions for matching comparison. Regions in the same stratum hold an identical credit, which is used for similarity metric. For the sake of matching flexibility, a dynamic adjustment scheme of credit setting is offered. In this sense, a user can arbitrarily adjust the credit parameters to pursue better retrieval results. Besides, a peak inspection technique is also added in the QMS matching metric to enhance performance. This means can helpfully refine retrieval performance with trivial computational cost. Experimental results reveal that effectiveness and efficiency of QMS are comparable to the Motif Cooccurrence Matrix (MCM) method while QMS is competent to deal with image scaling.

Keywords

Content-based image retrieval, motif scan, quadrant segmentation

1. INTRODUCTION

Visual information has proliferated for multiple purposes in recent years. With the Internet burst, a number of multimedia applications are evolving pervasively for intuitive information expression. Over the decades, many brilliant researches have produced a variety of outstanding techniques in image-related fields, and some of those became the de facto standards, such as JPEG [Pen93]. However, those mature studies largely reside in image encoding and storage format. By contrast, a wide range of approaches [Mah03, Pou04, Swa91, Pas96] using color, shape, or other factors for *Content-based image retrieval (CBIR)* is still under sprightly development. They retrieve images on particular occasions while some existent CBIR systems provide users with versatile querying ability [Nil93].

There are a plenty of works in the image retrieval area. Some of the renowned approaches are recognized as introductory examples for later works. For instance, color histogram [Swa91] is one of those precedents. It utilizes the statistics of global color distribution to calculate the similarity between two images. Afterwards, other advanced techniques were devised to remedy color histogram afterwards. Color Coherence Vectors (CCV) [Pas96] adds the information of pixels coherence of one particular color and generates coherent regions. This enhances distinction of pixels with same colors but not distributed in the same regions. Moreover, another attractive technique, Color Correlogram (CC) [Hua97], highlights the spatial correlations of colors. It takes on the probability of joint occurrence from any two pixels in separate colors or an identical color for autocorrelogram [Hua97]. Both of the two methods appear to perform much better than traditional color histogram. In a word, they presented the significance of spatial information in image retrieval. Not only the mentioned methods but more related studies have shown spatial property feasible and useful to retrieval refinement. From this point of view, we adopt the spatial factor into our work to reduce fidelity loss. Other schemes like Blobworld [Car02] and SIMPLIcity [Wan01], region-based techniques for image retrieval flourish in an alternate way.

Jhanwar et al. [Jha04] use motif notion to capture the low level semantics of space filling curves. Their *Motif Cooccurrence Matrix* (MCM) is the container

FULL Papers conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

of motif features literally scanned from image pixels. Each 2 × 2 pixel grid is replaced by one from a set of six Peano scan motifs [Pea90, See97]. This particular version of motifs is shown in Fig. 1. For each cooccurrence, it accumulates times of finding a motif *i* at a distance *k* from a motif *j* in the matrix. Efficiently, it merely includes individual 6 × 6 motif matrices for every color plane irrespective of the image size. Apparently, the MCM scheme is especially suitable for retrieving images in equal size. Variations in image size can inevitably induce inconsistent motif amount so it may lead to a meaningless matching process. Meanwhile, because the granularity of motif san is so minute, this is beneficial for textured images. Once the query scenario is in images from outdoor scene, the performance might drop down unexpectedly.

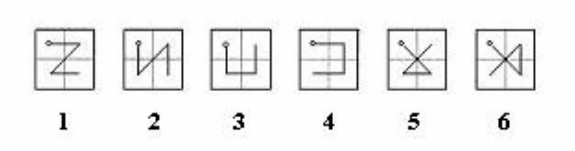


Figure 1. Codes for each type of motifs

In this paper, we propose another image retrieval approach based on motif symbols, called *Quadrant Motif Scan* (QMS). Consecutive quadrant segmentation on images is the main strategy in our scheme. With a motif extraction for each region, we collect all motif data for matching. By the hierarchical structure of motif information inside an image, we devise a matching algorithm for similarity comparison with ranked results. Our experiments show that QMS has the merits comparable or even superior to the MCM method.

This paper is organized as follows. In section 2, the proposed QMS is explored. Section 3 describes the matching metrics used in QMS. In section 4, the experimental results are presented with a related discussion. The final section gives the conclusion and prompts the relevant errands for future work.

2. QUADRANT MOTIF SCAN

Motif is an important feature to express the chromatic trend lying in a bounded region. In the QMS scheme, we use an entire image as the initial region, as shown in Fig. 2a. An image is first subdivided into four non-overlapping parts in Fig. 2b. The four segmented quads form the source generating a motif for the first region which also belongs to the 1st stratum. In Fig. 3, QMS then calculates separate mean values of all pixels for each quad and uses them to derive a motif. Here, the RGB colorspace is applied and the 3 colors are synthesized by averaging them, ranging from 0 to 255.

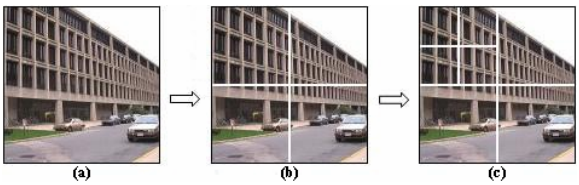


Figure 2. The recursive segmentation processes in Quadrant Motif Scan. (a) The original image. (b) 4 quads for the region in the 1st stratum. (c) 4 quads for successive sub-region.

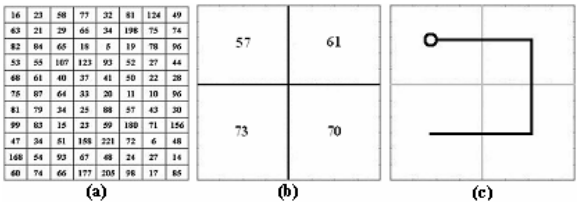


Figure 3. Formation of a motif from a region. From left to right are (a) image pixel values, (b) means of four quads, and (c) a resulting motif.

Likewise, successive subdivision operations (see Fig. 2c) from the current region continue until a pre-defined stratum threshold is reached. Continually, the same manipulation to evaluate mean values is carried out for every child region, and separate motifs are eventually derived as shown in Fig. 4. In particular, the four motif blocks belong to the same stratum (2nd) so they share a common credit for the matching metrics. In short, a parent's quad, used as an element for its motif derivation, is regarded as a child's intrinsic region.

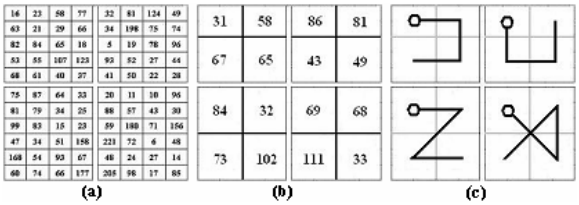


Figure 4. Successive subdivisions into four child regions.

To avoid ambiguity of motif scan, we introduce the suggestion used in the MCM study [Jha04]. Our QMS follows the breadth-first strategy to regulate the motif recognition. For instance, Z-type motif is recognized rather than N-type in Fig. 5. Note that any kind of motif traverse starts from the upper-left

quad regardless of its actual value. Besides, supplementary uniformity detection is taken into account simultaneously. During a motif extraction, the highest and lowest mean values from four quads are recorded. A simple subtraction operation is performed to retain the information whether the region is uniform or not. Finally, both of the motif and uniformity information for a region is collected and assembled together into a single data structure. The uniformity threshold in our scheme is set to 35 in default; the difference less than the threshold will lead a region to be uniform.

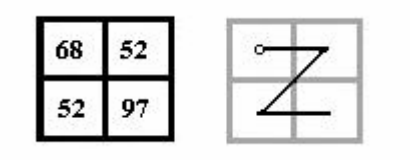


Figure 5. Example of motif traverse.

This architecture of stratified motif scan is to find out local motif information throughout an image. Due to different significance of separate strata, varied credits, or so-called weights, are granted to reflect their power. In the QMS scheme, we designed an algorithm using these credits to match images. On the other hand, what the exact stratum threshold is supposed to be becomes another issue. If a higher stratum threshold is designated, more blocks of motif are then received and can depict an image more precisely. From this viewpoint, the retrieval performance can explicitly rise via adding more strata at the expense of computational costs. In practice, an adjustable stratum threshold may be needed according to the user's demand. As a general rule, performance choice is recommended to come to a compromise between speed and effectiveness. Table 1 shows a cross mapping among strata, regions, and motif blocks.

Stratum No.	1	2	3	4	5	6	7
Regions	1	4	16	64	256	1024	4096
Accumulated motif blocks (A)	1	5	21	85	341	1365	5461

Table 1. Variations in corresponding numbers of strata, regions, and motif blocks

3. MATCHING METRICS

Before matching two images, there are some prerequisite preparations to launch image comparison.

System Settings

Stratum Threshold - A pre-defined stratum threshold is set for the motif scan process. All images are manipulated by this rule so that they have the consistent number of motif blocks stored in the database for the matching functionality.

Uniformity Criterion - The method requires a criterion for uniformity detection. This is used as a value range to recognize whether a region/motif block is uniform or not. The same as stratum threshold, this must be decided at the very beginning.

Credit Setting - Varied credits (weights) are specified for each stratum. These variables are in relation to retrieval results.

The first two items are the most important parameters for QMS implementation. Both of them can also be critical to retrieval performance. In turn, the third setting is about scoring schemes and is independent of the two previous settings. This can be dynamically adjusted at run-time to evaluate and rectify the retrieval results.

Distance

Assumed that the required parameters are verified and the QMS database is then constructed, the matching process can launch on the final comparison stage. Motif blocks of two images, the query and target, are matched on the blockwise basis. If a pair of corresponding blocks/regions is exactly the same, which have equal motif type and uniformity property, the target scores the specified credit. Otherwise, the target fails in this block and goes on next until all motif blocks are through. When the process is finished, a total score will be attached to that target. The higher the score is, the more similar the query and target images are. In effect, it will receive a perfect score when the query and target are exactly the identical image. After all images in the database are thoroughly matched with the query, a score list is made in a descending manner.

Given a mapping function, it can find out the *j*th stratum to which the *i*th block belongs and hence the corresponding credit $\boldsymbol{C_j}$. Each pair of motif blocks, M_i^q and M_i^t, for the query and target respectively, is compared to get a similarity status $\boldsymbol{S_i}$. If two blocks are equal, it leads to $\boldsymbol{S_i}$ = 1, or 0 otherwise. The overall matching expression is as follows:

$$D(I_q, I_t) = \sum_{i=1}^{K} S_i C_j \quad \forall j \in [1 \ldots N],\ j = map(i)$$

$$with\ S_i = \begin{cases} 1, if\ M_i^q = M_i^t \\ 0, if\ M_i^q \neq M_i^t \end{cases}, \mathrm{A}[j-1] < i \leq \mathrm{A}[j], \qquad (1)$$

where I_q and I_t are the query and target images and K means the maximum of motif blocks assigned by the number of strata N specified for motif scan process. Array A gives the information the ith block is subsidiary to the jth stratum. This distance metrics will result in a score sum for ranking.

Besides the parameters in motif construction, we add a peak inspection function for retrieval supplement. This function uses a peak number to set how many global peaks of pixel values in an image are selected. These peaks are traversed from the highest with proximal peaks omission. To enhance the proposed QMS method, the inspection will result in a percentage of matched peaks between images and use it to weight original matching score. The computational cost is trivial to this additional enhancement in exchange for advancement of precision.

Scheme	Strata	Motif Blocks	Uniformity	Peaks
A	5	341	35	10
B	6	1365		

(a)

Stratum No.	1	2	3	4	5	6
Credit	4	4	4	4	4	4

(b)

Table 2. (a) QMS system parameters and (b) Credit settings for scoring

4. EXPERIMENT RESULTS

Query-By-Example is a common technique used in many image retrieval systems. For instance, IBM QBIC [Fli95] is one of the best-known systems. The QMS system also applies such a skill in the system design. On the side, we chose the MCM method to serve as the opposite to show the comparison of motif-based approaches.

Experimental Setup

For a variety of queries, we collected a huge amount of images to test the retrieval performances in different query scenarios. These resources includes: nearly 10,000 images used in the WBIIS system [Wan98], 25 images from the Aridi art collection, and the MIT Media Lab's Vistex collection. Textured and non-textured images are existent in our experiments for specific comparison purposes. Scaling manipulation for some samples is also done to emphasize that QMS is invariant to image scaling. About the stratum threshold, it is adequately set to 5 in default with 341 motif blocks as a good start point. This setting is, however, still subject to variations of image size. Table 2 shows the related parameters used in our experiments.

With different setups, two typical schemes (see Table 2a) are presented for performance analysis. Scheme B is used to seek if there is a better solution after scheme A is tried out. In most cases, one more stratum (adding more motif blocks) could bring out substantial retrieval effectiveness. However, some other factors more or less give rise to direct or indirect influences on our multi-stratum comparison.

Again, the credit settings in Table 2b are for examples and could be dynamically adjusted in our experiments. Since there may be many diverse images in the database, a universal credit setting for all sorts of queries is really hard to define. We therefore design this flexible facility to meet needs in possible demands. This feature is also very helpful when we want to clarify an empirical credit setting toward a specific content in a query image.

Results

4.2.1 Database from MIT Vistex

We have performed voluminous queries in order to identify which type of images is more favorable for the QMS scheme. The 128 × 128 set of images in the MIT Vistex database was the first case in our comparison with the MCM method. Information for the classification and corresponding quantities in the database is shown in Table 3.

Fig. 6a shows the query image, and the retrieval results in Fig. 6b expose the effectiveness of the MCM method, retrieving 10 out of 11 in the series of buildings. Although the results may cause different

Class	Bark	Brick	Buildings	Fabric	Flower	Food	Grass	Leaves	Metal
Pieces	13	9	11	20	8	12	3	17	6

Class	Misc.	Paintings	Sand	Stone	Terrain	Tile	Water	Wood	Total
Pieces	4	13	7	6	11	11	8	3	162

Table 3. Image classification in the 128 × 128 set from the MIT Vistex database

evaluations from individual perceptions, some characteristics are apparent to realize. For example, the intricate grids, which form the appearance of the buildings, are the most salient features in this query image. On these grids, two colors black and yellowish gray are alternate regularly throughout the surface of the buildings. The extent of buildings also occupies most of the image. Hence, we can intuitively infer that a great deal of some repeated motifs will be extracted and stored in the MCMs. Because of the subtle granularity to form a motif, any massive prominent patterns are crucial to retrieval. Despite the fact that there are a few deficiencies in this retrieval sample, the performance is brilliant. However, queries in other categories in Table 3 do not necessarily give the same performance as well.

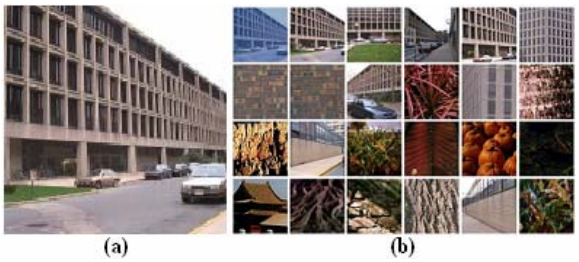


Figure 6. (a) The query image and (b) retrieval results using the MCM method. The retrieved images are ordered from left to right and top to bottom by their similarity to the query image.

Referring to the performance measure in Guoping Qiu's study [Qiu03], we made a slight modification of it and only the first 24 retrieved images in the first page in our system are shown for discussion. Other images are still ranked and remained below the retrieval manifest.

The same query sample is conducted in QMS as shown in Fig. 7. We use two schemes, A for 5 strata with the credit setting 4-4-4-4-4 and B for 6 strata with 4-4-4-4-4-1, to demonstrate performance. In scheme A, the results in the first row seem better than MCM, but, in terms of quantity, A merely retrieves 5 images of the same category (buildings). In turn, B improves both retrieval quality and quantity from A. Obviously, scheme B outperform MCM in retrieval quality, i.e. the ranking order, with a little inferior quantity of 8. At least, from synthesizing both criteria, our QMS remains quite comparable.

Furthermore, we made an interesting experiment to evaluate retrieval ability when the database only consists of images in the buildings category. As we can see in Fig. 8, the similarity order is much more reasonable in QMS than that of MCM. Thus, this phenomenon can explain that MCM performs considerably well while desired images in database are perceptibly distinct from others. Otherwise, ordinary queries may not give such a satisfactory performance as the previous example. Moreover, another query scenario is reported in [Lin05] when there is only one image in database perceptually similar to the query.

Figure 7. Retrieved images by QMS with different schemes. The stratum threshold is set to 5 in A and increased to 6 in B.

Figure 8. Results from querying only in the buildings category by (a) MCM and (b) QMS.

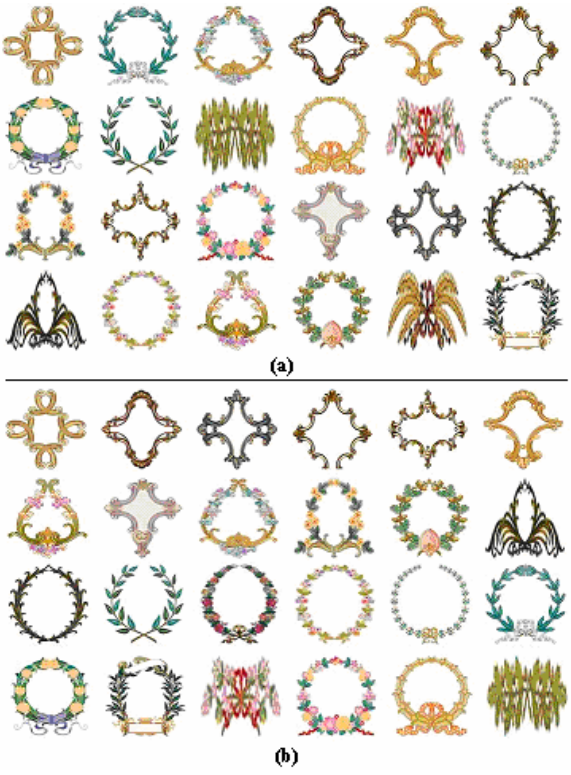


Figure 9. Retrieval results from images of unequal sizes by (a) MCM and (b) QMS. All images are displayed in thumbnail.

4.2.2 *Querying in images of unequal sizes*

Next, the experimentation will proceed to databases containing images of unequal sizes. To account for the retrieval capability invariant to image scaling, a set of 25 color images was collected from the Aridi art collection. These images differ in size ranging from 239 × 363 to 724 × 192 in disproportional ratios. A sole entity located in the center, such as a crest or ribbon, is the common property of the images. From the query example in Fig. 9a, the performance of MCM obviously drops down due to unfixed image sizes; it can not match images on an inconsistent criterion. In effect, the match results almost lie in a disordered state. By contrast, along with the setting of scheme A in Table 2 (without peak inspection), QMS make the ranking orders in Fig. 9b much more acceptable than MCM. Ideally, images with a cross should be retrieved first before those with a circle. In this context, QMS almost fulfills this requirement. Therefore, our approach shows the capability of querying in which target images vary in size. In another trial, however, QMS is still sensitive to image rotation, whereas the MCM might be less sensitive.

4.2.3 *Performance evaluation*

As mentioned in the study of Müller et al. [Mul01], evaluation measures to retrieval performance are proposed from various viewpoints. Similarity judgments are not easily be done objectively. Thus, we resort to the notion of *recall* in this paper and modify the formula as follows:

$$\text{recall}\ (M_1) = \frac{\|\text{First}_m(M_1) \cap U\|}{m}, \qquad (2)$$

where $\text{First}_k(M)$ is the set of the first k retrieved images by the method M. $U=\text{First}_k(M_1) \cup \text{First}_k(M_2)$, $m = \|U\|$, i.e., m is the number of members in the set U while methods M_1 and M_2 are MCM and QMS in our experiment respectively. If there are many methods, M_1, M_2, ..., M_n, to be evaluated, then only the definition of U in this formula must be changed, i.e., $U = \text{First}_k(M_1) \cup \text{First}_k(M_2) \cup ... \cup \text{First}_k(M_n)$.

The first evaluation is for queries conducted from the database in Table 3. In this process, we randomly picked out 50 images (total 162) as the query and made k equal to 20 to gain the first 20 results separately from MCM and QMS. Then, an intersection (m) is made as the denominator. Besides, we repeated this process for at least 3 times to avoid biased evaluations. Table 4 shows the average times of better recall values. The schemes in use conform to the credit setting in Table 2b.

In Table 4, 5S means using 5 strata in the QMS while 5SP is with auxiliary peak inspection. TIE counts the times where the recall values of both methods are equal. From this table, increment of stratum can help raise the recall evaluation for QMS, and so does the incorporation of peak inspection. In Addition, we also apply this performance measure to the database from WBIIS (see Table 5). In this context, images are largely heterogeneous so that the accumulated times of TIE are greater than any of the other two. This situation implies that most retrieval results by MCM and QMS are not overlapped. Hence, our QMS is not arguably better than the other though the data is advantageous to QMS. From the statistical data in Table 4 and 5, however, QMS is proven superior to MCM based on this recall criterion.

	Average Times		
Schemes in QMS	MCM	TIE	QMS
5S	20	13	16
5SP	15	14	21
6S	21	7	22
6SP	15	12	23

Table 4. Average times of better recall values

	Average Times		
Queries	MCM	TIE	QMS
300	63	140	97
3000	283	2369	348

Table 5. Average times of better recall values from the database used in WBIIS

Through a plenty of experiments, QMS is very efficient to computation and economical to storage requirement. In those experiments, we use a tentative platform based on PC, a Pentium III 733MHz CPU with 512MB RAM. The time costs of motif feature construction in Table 4 are approximately 7 sec for MCM and QMS while both use less than 1 sec in matching images. For the second case in Table 5, both methods spend approx. 5 min on offline motif extraction for nearly 10,000 images while QMS merely uses less than 3 sec for matching, which is better than 11 sec of MCM. Apparently, QMS can much better fulfill the speed requirement (in linear time) of online query. Note that all time data above settles on a temporary file database system.

5. CONCLUSIONS

In this paper, we have introduced another retrieval approach, Quadrant Motif Scan, based on motif

symbols. Images are segmented recursively into sub-regions, which are the sources of motif derivation. Motif scan is limited to a stratum threshold and combined with uniformity detection. With any arbitrary or empirical credit setting, the matching metrics is conducted in a simple, fast manner. Incorporating the additive of peak inspection, QMS may run at a more stable and reliable performance. Compared with the MCM method, QMS remains competitive and even better, especially when image sources are not highly textured. With the additive of peak inspection, the QMS can run at a more stable and reliable performance. Most important, the mechanism of relevance feedback is provided in the QMS system. Users can dynamically adjust the credit setting to achieve optimal results. Nevertheless, the problem of image rotation remains awkward to overcome in QMS.

Motif is a descriptor capturing features for images. Although it is not comprehensive for all features about images, it is applicable to employ with other algorithms or advanced techniques. So far, efficiency and effectiveness are demonstrated in the current method to a certain extent. In the future, we might seek to develop better algorithms to complement this motif-based method. Empirical rules of credit setting for diverse query scenarios are required analyzing in detail. In future work, we will count on other considerations like colorspaces, e.g. HSV, for further reinforcement.

6. ACKNOWLEDGEMENTS

We are indebted to fellows in the lab for many constructive opinions. This work is funded by the Taiwan NSC under Grant No. 94-2213-E-031-006.

7. REFERENCES

[Car02] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik, "Blobworld: A System for Region-Based Image Indexing and Retrieval," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 8, pp. 1026-1038, Aug. 2002.

[Fli95] M. Flickner et al., "Query by Image and Video Content: The QBIC System," *IEEE Computer*, vol. 28, no. 9, pp. 23-32, Sept. 1995.

[Hua97] J. Huang et al., "Image Indexing Using Color Correlogram," in *Proc. CVPR97*, pp. 762-768, 1997.

[Jha04] N. Jhanwar et al., "Content Based Image Retrieval Using Motif Cooccurrence Matrix," *Image and Vision Computing*, vol. 22, pp. 1211-1220, 2004.

[Lin05] T.W. Lin and C.S. Hung, "Content Based Image Retrieval Using Quadrant Motif Scan," *Proc. IEEE Int'l Conf. on Systems, Computing Sciences and Software Engineering*, in press.

[Mah03] F. Mahmoudi et al, "Image Retrieval Based on Shape Similarity by Edge Orientation Autocorrelogram," *Pattern Recognition*, vol. 36, no. 8, pp. 1725-1736(12), Aug. 2003.

[Mul01] H. Müller et al., "Performance Evaluation in Content-based Image Retrieval: Overview and Proposals," *Pattern Recognition Lett.*, 22, pp. 593–601, 2001.

[Nil93] W. Niblack et al., "Querying Images by Content Using Color, Texture and Shape," *Proc. SPIE*, vol. 1908, pp. 173-187, 1993.

[Pas96] G. Pass, R. Zabih and J. Miller, "Comparing Images Using Color Coherence Vectors," *Proc. ACM Conf. on Multimedia*, pp. 65-73, 1996.

[Pea90] G. Peano, Su rune courbe qui remplit toute une aire plaine, Mathematicshe Annalen 36 (1890) 157-160.

[Pen93] W.B. Pennebaker and J.L. Mitchell, "JPEG Still Image Data Compression Standard," Van Nostrand Reinhold, New York, 1993.

[Pou04] H. Nezamabadi-pour and E. Kabir, "Image Retrieval Using Histograms of Uni-color and Bi-color Blocks and Directional Changes in Intensity Gradient," *Pattern Recognition Lett.*, vol. 25, pp. 1547-1557, 2004.

[Qiu03] G. Qiu, "Color Image Indexing Using BTC," *IEEE Trans. Image Processing*, vol. 12, 2003.

[See97] G. Seetharaman, B. Zavidovique, Image processing in a tree of Peano coded images, in: Proceedings of the IEEE Workshop on Computer Architecture for Machine Perception, Cambridge, CA, 1997.

[Swa91] M. Swain and D. Ballard, "Colour indexing," *Int'l J. Computer Vision*, vol. 7, pp. 11-32, 1991.

[Wan01] J.Z. Wang, J. Li, and G. Wiederhold, "SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture LIbraries," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 23, no. 9, pp. 947-963, Sep. 2001.

[Wan98] J.Z. Wang, G. Wiederhold, O. Firschein, and X.W. Sha, "Content-Based Image Indexing and Searching Using Daubechies' Wavelets, " *Int'l J. Digital Libraries*, vol. 1, no. 4, pp. 311-328, 1998.

Internal and External Salient Points under Affine Transformations. Comparative Study

S. Mashtalir
Department of Computer Science
Kharkov National University of Radio Electronics
Lenin ave. 14
61166, Kharkov, Ukraine
mashtalir_s@kture.kharkov.ua

K. Shcherbinin
Department of Computer Science
Kharkov National University of Radio Electronics
Lenin ave. 14
61166, Kharkov, Ukraine
konstantin@inetvisor.com

E. Yegorova
Department of Computer Science
Kharkov National University of Radio Electronics
Lenin ave. 14
61166, Kharkov, Ukraine
yegorova@kture.kharkov.ua

ABSTRACT

Permanently a special emphasis is given to image processing considering geometric distortions for remote sensing, and image tracking especially. We discuss an approach to affine transformation parameters search on the base of salient points which can be external and internal relative to shapes. To find affinity we use sets of image binary cuts and one-to-one point correspondence. This correspondence is determined by means of the affine-equivalent shape ratio invariance. We investigate two types of salient points on the base of binary morphology. First ones are vertices of convex hulls and second ones belong to skeletons. Finally, solution of overdetermined system of linear equations gives us affine transformation parameters. Results of the exterior points and skeleton normalization methods comparative analysis are discussed.

Keywords

Salient points, mathematical morphology, image normalization.

1. INTRODUCTION

Invariance to natural and artificial deformations or their elimination represent one of the traditional pattern recognition problems, which still does not have a final solution. With reference to image processing it is typical to deal with geometric distortions induced by change of a mutual location of object and videosensor. In most cases distortions of this kind are simulated by affine transformations. Despite of numerous researches on normalization [see, e.g. Rot96, She99, Pei95] there are some classes of images for which it is expedient to synthesize new algorithms based on the shape feature analysis.

A shape analysis is most convenient applied to binary images. Nevertheless the valid acquisition of binary image is sufficiently challenging problem for arbitrary gray-level videodata. Reasonable approach is to process a whole set of image binary cuts simultaneously. The reduction of gray-level image processing to binary cuts sets analysis provides high accuracy and reliability of normalization and is rather simple for unified hardware-software implementation including SIMD architectures first of all. Thus the solution can be based on the image affine-equivalent salient points, which steadily characterize shape properties.

Steadfast attention is devoted to the points of interest (salient points) search [see, e.g. Cha97, Dau01, Mik04, Zhu95, Gol98, Spr94, Sud97]. Furthermore, traditional use of interactively indicated reference points should be mentioned [Dau01, Zhu95]. However in some cases human interference in image processing system operations is difficult or generally impossible. At the same time, known automatic procedures of affine-equivalent points search do not have a sufficient noise stability with reference to normalization problems.

Thus, our objective is to present and experimentally investigate reliable procedures of searching the image salient points and detecting one-to-one affine corresponding point (feature points) among them. We describe techniques of salient points (and

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

indexed subsets) search and experimentally analyze their effectiveness.

Indices of processing speed, accuracy and noise stability are used as effectiveness characteristics of normalization. Offered normalization methods are based on binary morphology operations [Ser82, Hei94, Gia88, Mar02] and lie in the automatic search of binary images (or cuts) affine-equivalent points. Two techniques of feature points search are investigated. They depend on external points and on image skeletons. The statistical analysis of image indexed salient points subsets search methods has been carried out to define specification of their practical use.

2. METHODS OF IMAGE AFFINE-EQUIVALENT POINTS SETS SEARCH

2.1 Normalization problem statement

Suppose that in some time poins t', t'' two images $\mathrm{B}'(x), \mathrm{B}''(x)$ ($x \in \mathrm{R}^2$) of the same object are fixed in videosensor field $\mathrm{D} \subset \mathrm{R}^2$ and they are connected by affine planar transformation $\mathrm{B}'(x) = g \circ \mathrm{B}''(x)$, i.e. $x \in \mathrm{D}, g \in \mathrm{G} = Aff(2, \mathrm{R})$. The group G action with identical transformation denoted by e has form

$$\mathrm{B}'(x) = \mathrm{B}''(\mathrm{A}x + b), \qquad (1)$$

where $\mathrm{A} \in Gl(2, \mathrm{R})$ is a nonsingular matrix of the second order, $b \in \mathrm{R}^2$ is a translation vector. Note, that each image can be represented as a union of m binary cuts (generally in mixed scale of notation)

$$\mathrm{B} = \bigcup_{i=1}^{m} \mathrm{B}_i w_i \ .$$

Here $\mathrm{B}_i = 1 \ \forall \gamma \in \mathrm{D}: \mathrm{B}(\gamma) \in [\alpha_i, \beta_i]$, α_i, β_i are given or found thresholds of gray levels partitions and $\mathrm{B}_i = 0$ otherwise, w_i are weight numbers connected to the image representation i.e. the radix number steps of gray levels can be chosen as their discriminant homogeneous regions (see Figure 1). It is obvious, that $\mathrm{B}'_i = g \circ \mathrm{B}''_i$.

The problem consists in searching image geometric transformation parameters $\mathrm{A} \in Gl(2, \mathrm{R})$, $b \in \mathrm{R}^2$ on given sets $\{\mathrm{B}'_i\}_{i=1}^m, \{\mathrm{B}''_i\}_{i=1}^m$.

Let us underscore that transfer to binary cuts analysis reduces a normalization problem only to proceeding shape characteristics of object image and its components, in particular its individual points. Further, we shall omit index i and consider one cut as a binary image B, identifying it, if necessary, with a gray-level image.

Figure 1. An image and binary cuts

Let us consider methods based on the salient points analysis. It is related to the fact that stable automatic selection of affine-equivalent points provides simple algorithmization of geometric transformation parameters search (i.e. normalization problem solution) as they are found straightly from a linear equations system

$$\begin{cases} x'_{1,k} = a_{11} x''_{1,k} + a_{12} x''_{2,k} + b_1, \\ x'_{2,k} = a_{21} x''_{1,k} + a_{22} x''_{2,k} + b_2, \end{cases} \quad k = \overline{1, \mathrm{K}}, \ \mathrm{K} \geq 3. \qquad (2)$$

Especially we shall mark, that accurate and reliable affine-equivalent points search alongside with its computational simplicity, like other local methods provides a number of additional possibilities e.g. partial objects overlapping consideration, removing the object from videosensor field, i.e.

$$\exists x \in \mathrm{D}: \mathrm{B}(x) \neq 0, \ \ \mathrm{A}x + b \notin \mathrm{D} \ .$$

2.2 Image salient points sets

Let us introduce the basic definitions necessary for binary image affine-equivalent points search and validate effectiveness of mathematical morphology methods application as theoretical and practical toolkit. First we shall suppose that a map belonging to the power map $\mathcal{F}: \mathrm{B} \to 2^{\mathrm{B}}$ is given and it extracts the affine-equivalent points families. Then (after the necessary properties are established) we shall find such maps.

Definition 1. For arbitrary binary image B' elements of set $\mathcal{F}(\mathrm{B}') = \{x\}$ we shall name salient points, if

$$\forall g \in \mathrm{G} \ \forall x' \in \mathcal{F}(\mathrm{B}') \ \ \exists! \, x'' \in \mathcal{F}(\mathfrak{B}'') : x' = g \circ x''. \qquad (3)$$

It is clear, that equation in (3) is also valid inversely

$$\forall g \in \mathrm{G} \ \forall x'' \in \mathcal{F}(\mathrm{B}'') \ \exists! \, x' \in \mathcal{F}(\mathrm{B}') \ : \ x'' = g^{-1} \circ x'.$$

As an example of set $\mathcal{F}(\mathrm{B})$ it is possible to specify boundary $\partial \mathrm{B}$ of image B. At first sight uniqueness

of the affine-equivalent points existence in (3) is rather obvious by virtue of the group transformations properties. However it is necessary to take into account, that images are actually considered not in $D \subset R^2$, but in $D \subset Z^2$, what may lead to violation of the group transformations properties, i.e. from the equality $\forall x \in Z^2,\ g \circ x = x$ does not necessarily follow $g = e$, and also there may exist a subregion of the videosensor field $D^* \subseteq D$ such, that

$$\exists g \in G : B(x) \neq 0, \forall x \in D^* \Rightarrow Ax + b = const$$

Sufficient illustration can be a scaling, when one boundary ∂B point can correspond to several points under same transformation parameters.

Definition 2. The indexed salient points $\Theta \subseteq \mathcal{F}(B)$, $\Theta = \{\theta_i\}_{i=1}^{I}$, $I \leq card\ \{\mathcal{F}(B)\}$ we shall name feature points, if $\forall B \in M$, $\forall g \in G$ there is a permutation $\pi_{\mathcal{F}} = \begin{pmatrix} 1 & 2 & \dots & I \\ \alpha_1 & \alpha_2 & \dots & \alpha_I \end{pmatrix}$ such, that

$$\theta_i = g \circ \theta_{\alpha_i},\ \theta_i \in \mathcal{F}(B),\ \theta_{\alpha_i} \in \mathcal{F}(g \circ B).$$

Directly from the definition it follows that condition $card\ \{\mathcal{F}(B)\} = card\ \{(g \circ B)\}$ or more precisely $card\ \Theta = card\ g \circ \Theta$ should be satisfied. In other words, it is necessary that map $\mathcal{F}$ would not depend on planar affine transformations.

When we select n feature points and form matrices

$$\Lambda' = \begin{pmatrix} \theta'_{x1} & \theta'_{x2} & \dots & \theta'_{xn} \\ \theta'_{y1} & \theta'_{y2} & \dots & \theta'_{yn} \end{pmatrix}$$

$$\Lambda'' = \begin{pmatrix} \theta''_{x1} & \theta''_{x2} & \dots & \theta''_{xn} \\ \theta''_{y1} & \theta''_{y2} & \dots & \theta''_{yn} \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

required affine parameters are

$$(A\ b) = \Lambda'\Lambda''^{T}(\Lambda''\Lambda''^{T})^{-1}.$$

Let $B_\phi \subseteq B, \phi \in \Phi$, we shall impose on producing salient points map $\mathcal{F} : B \to 2^B$ conditions

$$\mathcal{F}(\varnothing) = \varnothing,\ \forall B_\phi \subseteq B : B_\phi \neq \varnothing \Rightarrow \mathcal{F}(B) \neq \varnothing, \quad (4)$$

$$\mathcal{F}(\bigcup_{\phi \in \Phi} B_\phi) = \bigcup_{\phi \in \Phi} \mathcal{F}(B_\phi), \quad (5)$$

$$\forall B', B'' \subset B : B' \subset B'' \Rightarrow \mathcal{F}(B') \subset \mathcal{F}(B''), \quad (6)$$

$$\mathcal{F}(\bigcap_{\phi \in \Phi} B_\phi) \subset \bigcap_{\phi \in \Phi} \mathcal{F}(B_\phi). \quad (7)$$

Condition (4) specifies existence of the required map. The condition of additivity (5) together with a condition of inheriting (6) guarantees a possibility of fragment processing, i.e. creates premises for realization on SIMD-architecture. Condition (7) ensures processing of several images binary cuts, including outcomes of their interim processing. If $\mathcal{F}$ is a biunivocal mapping, inclusion in (7) grades into equality.

Let us search maps $\mathcal{F}$ by constructing set-theoretic operations superpositions and Minkowski algebra operations. It follows from the clear fact that the class of bit cuts is closed concerning Minkowski algebra operations and Minkowski addition and subtraction

$$X \oplus Y = \bigcup_{y \in Y} X + y,\ X \ominus Y = \bigcap_{y \in Y} X + y,$$

(X, Y are arbitrary sets) satisfy conditions (4)–(7).

We shall emphasize that if fixing one of the sets, namely, its spatial form and structure in operations (8), (9), we can receive subsets with the given properties in a range of definition. Constructive and traditional way of Minkowski algebra operations realization with reference to image processing binary morphology operations [Ser82, Hei94].

Let still B be a binary image defined in videosensor field $D \subset Z^2$. Basic operations are: dilation, coinciding with Minkowski addition

$$\delta_H(B) = \bigcup_{h \in H} (B+h) = \{x \in B : [(H + x) \bigcap B] \subset B\} \quad (8)$$

and erosion

$$\varepsilon_H(B) = \bigcap_{h \in H} (B+h) = \{x \in B : (H^r + x) \subset B\} \quad (9)$$

where $H^r = \{-x \in B : x \in H\}$, $H \subseteq D$ is a fixed set named as a structural element. We used structural elements H^* which are shown in Figure 2.

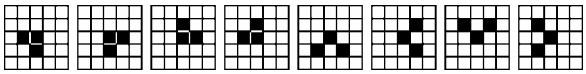


Figure 2. Set of structural elements

Sets of image affine-equivalent points can be of two types: interior and exterior. Interior points are related to images convex hulls, and exterior – to the centers of gravity, centroids, etc. For the considered two algorithms of interior and exterior feature points we used notations $N_8, N_4 \in H$:

$$N_4(\alpha) = \{\beta \in \mathbb{Z}^2 : |\alpha_x - \beta_x| + |\alpha_y - \beta_y| = 1\},$$

$$N_8(\alpha) = \{\beta \in \mathbb{Z}^2 : max\ \{|\alpha_x - \beta_x|, |\alpha_y - \beta_y|\} = 1\},$$

Then we can find boundaries

$$\partial_4(B) = B / \varepsilon_{N_8}(B),\ \partial_8(B) = B / \varepsilon_{N_4}(B)$$

and using combination of morphological operations (8) and (9) affine-equivalent skeleton was

constructed for the interior feature points set search

$$\Phi_i(\mathrm{B},\mathrm{H}) = (\varepsilon_{\mathrm{H}*}(\mathrm{B}_i / \varepsilon_{N_4}(\mathrm{B}_i)) \cup (\mathrm{B}_{i-1} / \delta_{N_8}(\mathrm{B}_i)), \quad (10)$$

where $\mathrm{B}_0 = \mathrm{B}$, $\Phi_0(\mathrm{B}, \mathrm{H}) = \varepsilon_{\mathrm{H}*}(\partial_8(\mathrm{B}_0))$, $\mathrm{B}_i = \mathrm{B}_{i\text{-}1} / / \partial_4(\mathrm{B}_{i\text{-}1})$, $i = \overline{1, \mathrm{K}}$, K is such that $\mathrm{B}_\mathrm{K} = \varnothing$, then the aggregate of points obtained with the help of (10) forms the map $\mathcal{F} = \bigcup_{i=0}^{\mathrm{K}} \Phi_i(\mathrm{B}, \mathrm{H})$ which produces the salient points forming affine-equivalent skeleton $Sk(\mathrm{B})$. Obtained in this way skeleton became a basis of the interior points binary image normalization method.

The second offered algorithm is based on the external points search. It can be proved that the following morphological operations combination allows to find external salient points set

$$\mathcal{F} = \varepsilon_{\mathrm{H}*}(\partial_8(\mathrm{B})) \quad (11).$$

As algorithm, described by (11), searches for the salient points, feature points set needed to be obtained by building an image convex hull, which peaks formed the required set. For affine-equivalent skeletons (10) node points (point which have more than two neighbors according to 8-connectivity paradigm) will be the feature points. To find correspondence between feature points sets of template and distorted images it is necessary to use property of areas ratio invariance under affine transformations. For the algorithm based on external point search we find permutation

$$\pi_{\mathcal{F}} = \begin{pmatrix} p & mod_n(p+1) & \dots & mod_n(p+n-1) \\ \alpha^* & mod_n(\alpha^*+1) & \dots & mod_n(\alpha^*+n-1) \end{pmatrix},$$

where p is a fixed feature point on the convex hull of image B_i' and α^* is the corresponding point of the convex hull of B_i'', n is the number of feature points. For internal feature points we proceed by analogy.

The point set obtained by means of such map has laid the foundation for the feature points binary images normalization methods.

However the question arises as to these methods accuracy, reliability and stability. With the purpose of these characteristics clarification, and also to determine their practical applications, we shall carry out the comparative analysis of these morphological normalization methods.

3. COMPARATIVE ANALYSIS OF MORPHOLOGICAL NORMALIZATION METHODS

Let us introduce the affine transformation group operation $Aff(2, \mathrm{R})$ (1) in homogeneous coordinates

$$\mathrm{B}'(x) = g \circ \mathrm{B}''(x) \sim \mathrm{B}'(x) = \mathrm{B}''(\mathrm{H}x),$$

where $x = (x_1, x_2, 1)^{\mathrm{T}}$, $\mathrm{H} = \begin{pmatrix} \mathrm{A} & b \\ 0 \;\; 0 & 1 \end{pmatrix}$. For realization of authentic statistical researches the knowledge of affine transformation true parameters and the values found by means of feature points is necessary. In other words, we shall obtain affine-equivalent images from templates $\mathrm{B}_0^k(x)$, $k = 1, 2, \dots$ using affine transformation, defined by 3×3 matrix H_t, i.e. $\mathrm{B}'(x) = \mathrm{B}_0(\mathrm{H}_t x)$. We shall denote obtained values as H_v, then the normalization error in a feature space will be

$$\varepsilon_{\mathrm{M}} = \sqrt{tr\,[(\mathrm{H}_v - \mathrm{H}_t)(\mathrm{H}_v - \mathrm{H}_t)^{\mathrm{T}}]}, \quad (12)$$

where $tr\,[\circ]$ is a spur of a matrix. We shall mark that given estimation can be used only if the normalization accuracy is high enough, as affine transformations are non-isometric.

Error in images space we shall define as measure of intersection of template $\mathrm{B}_0(x)$ and normalized $\mathrm{B}_v(x) = \mathrm{B}'(\mathrm{H}_v^{-1} x)$ images in the videosensor field

$$\varepsilon_{\mathrm{I}} = 1 - \frac{card\,(\mathrm{B}_0 \cup \mathrm{B}_v) - card\,(\mathrm{B}_0 \cap \mathrm{B}_v)}{card\,(\mathrm{B}_0 \cup \mathrm{B}_v)}. \quad (13)$$

In recognition problems the estimation (12) is more preferable; however at the experimental researches (when fixing some registration conditions and image processing) expression (13) gives a clearer understanding.

Under research of feature points normalization methods characteristics with the purpose of the binary cuts deriving algorithms influence elimination we shall use binary images only. 88 images used in a simulation modeling are shown in Figure 3.

Examples of normalization results using salient points and skeletons are shown in Figure 4 and Figure 5 accordingly.

On first column of Figure 4 one can see template images, on second one – distorted images. Feature points sets and convex hulls for template and transformed images are shown in columns 3 and 4 respectively. Last column contains normalization results obtained by method using (11).

In a similar manner in first two columns of Figure 5 template and distorted images are shown. Columns 3 and 4 contain skeletons for these images respectively and in the last column normalization results obtained by method using (10) are presented.

To determine whether the type of entering images has any influence on the normalization accuracy, i.e.

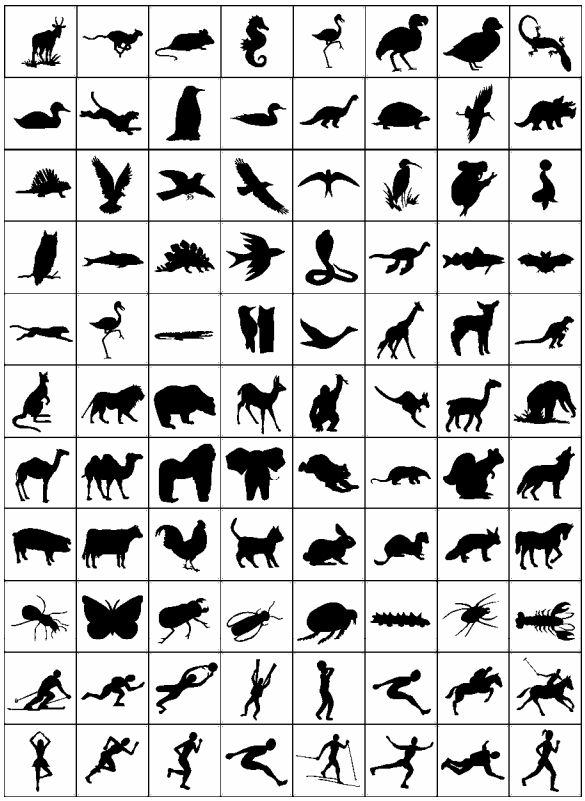


Figure 3. Binary images used in experiments

whether normalization averages are comparable, the variance analysis tools were used.

Let us note that independence of each sample from the others is provided by independency of processes image ensemble choice.

All images were introduced on the discrete raster having 512×512 pixels. To provide the equivalence of all images presentation in digital type (eliminations of quantization and digitization errors) they were obtained by converting various vectorized fonts versions of TTF format into BMP format files.

For variance analysis application it is necessary, that normalization errors would be distributed under the normal law and their variances are uniformal. To check the hypothesis of distribution normality, the computations results were broken into 10 intervals.

For each of them number of results m_i and probability of p_i hitting this interval under the

normal probability distribution law was determined.

Estimation of criterion value χ^2 at $l = 10$

$$\chi^2 = \sum_{i=1}^{l} \frac{(m_i - np_i)^2}{np_i} = 1.05,$$

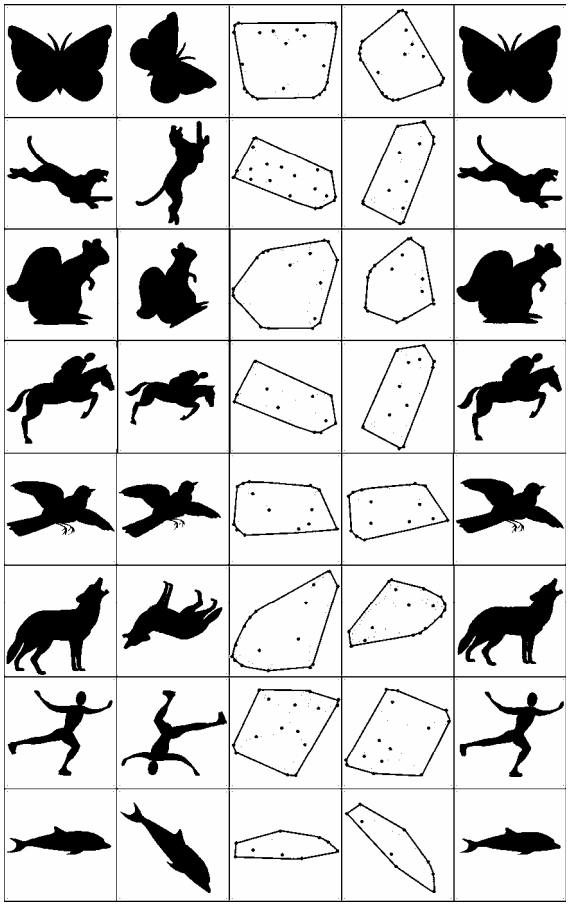


Figure 4. Normalization using exterior points

and critical criterion value under $k = 7$ degrees of freedom and $p = 0.99$ confidence probability is equal $\chi^2 = 1.24$, i.e. there is no sufficient foundation to reject a hypothesis about normal distribution.

Checking line uniformity hypothesis has provided the following results: choosing a confidence probability $p = 0.99$, we discover the upper critical limit χ^2, under number of degrees of freedom equal 87, $\chi^2 = 59.279$, i.e. we have accordingly

$$Q_1 = 21.74 < \chi^2 = 59.28,\ Q_2 = 4.02 < \chi^2 = 59.28,$$

what means that the hypothesis about variances line uniformity does not contradict the experiment results.

Further we shall calculate an unbiased estimator of deviations inside each series and get for the indicated data at degree of freedoms $k - 1 = 87$ and $N - k = 528$ accordingly

$$F_1 = s_{a,1}^2 \big/ s_{r,1}^2 = 1.63 > F_{0.01} = 1.43$$

and

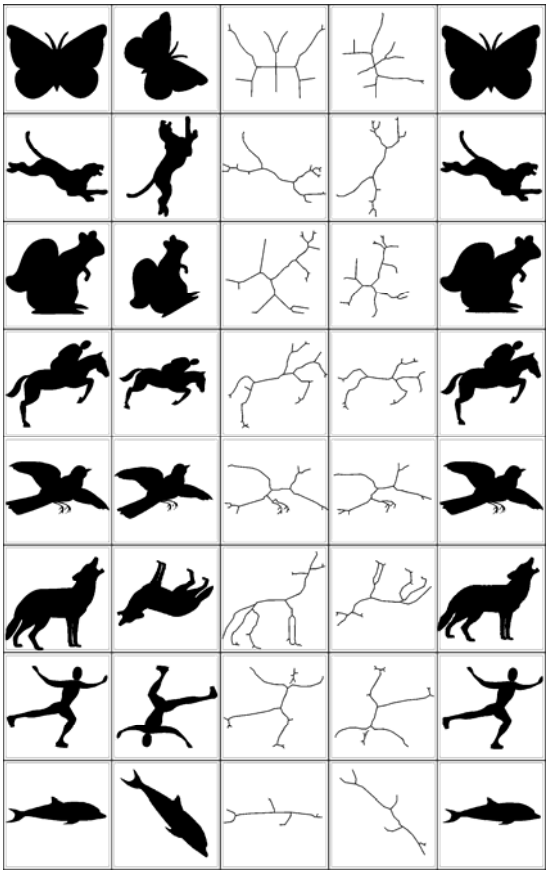


Figure 5. Normalization using interior points

$$F_2 = s_{a,2}^2 \big/ s_{r,2}^2 = 0.161 < F_{0.01} = 1.43 .$$

Thus, it is possible to state that based on the carried out experiments there are no foundation to consider that skeleton based normalization accuracy depends on the binary images type. At the same time, there is no reason to believe that extreme points morphological normalization does not depend on input images. The obtained result has a rather transparent explanation: external points extraction is not sufficiently steady at the small areas image fragments analysis, what brings the erroneous data into the simple equations systems. Elimination of such errors can be carried out by similar feature points analysis exclusion, and also by increasing the number of images binary cuts. On the other hand, skeletons construction is based on a local symmetry

of binary cuts, i.e. actually in an implicit aspect some integrated estimation of images is used, what ensures higher stability of the method. In other words, it is necessary to choose either faster or more reliable method with reference to applied specification of solving tasks, or use their combinations.

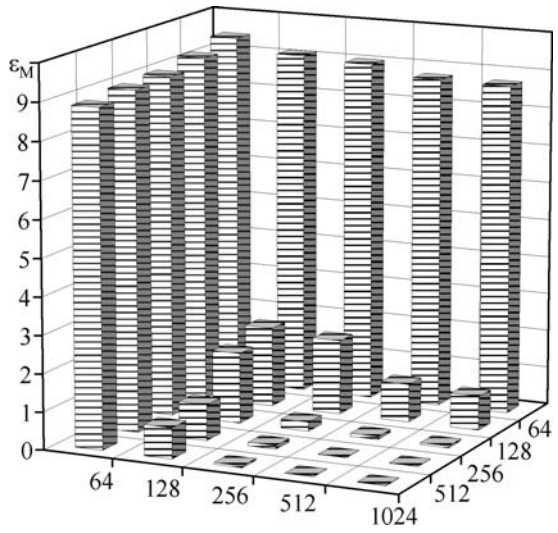


Figure 6. Averaged normalization error under different discrete raster

Let us underline, that discrete raster observation number also influences the end results. In Figure 6 averaged on 10 images and 5 affinities (50 experiments) skeleton normalization errors dependency on traditional for digital processing of videosensor field characteristics is shown. It is visible, that accuracy raises dramatically remains rather stable under observation/indication number exceeding 256.

Also experimental check of morphological normalization methods stability under conditions of noise presence was carried out. At first, the normalization error dependency from noise level and density in spatial area is considered. Gaussian and uniform noises were used as distortions. Following results were obtained: despite the density of noise, at the level up to 40 %, the normalization error in space does not exceed 2 % from the shape area, what is an acceptable result. Further, depending on a density it begins to increase considerably. The normalization error dependency from noise type was also considered. Results showed that normalization error in space changes more for uniform noise. Dependency from the noise level is much stronger than from their density.

Further the experimental analysis concerning geometric transformation parameters improvement on binary cuts sets was carried out. It is possible to draw a conclusion that under increasing of processed informative binary cuts number for the given class of images, the error of parameters determination decreases. However, if at the initial stage addition of new informative cut considerably reduces the average error for a class of images, starting with 6 cuts the error remains practically constant. From this

it is possible to draw a conclusion, that the optimal amount of processed informative cuts for considered classes of images is a number in a range from 3 to 6. It is necessary to note, that one should not misapply the amount of cuts since multiple increasing of their number will lead to the case when informativity of each single cut will decrease, what can lead to a significant error of image normalization parameters determination.

4. CONCLUSIONS

The basic scientific significance of this work is the new methods which describes image feature points sets on a discrete raster. Steady automatic search of those is one of necessary parts of image normalization problems solution, as used for geometrical transformation parameters search. We analyzed stability and reliability problems of searching the indexed salient points sets which are the solution of external points and skeleton morphology normalization tasks based on mathematical morphology operations combinations. For statistical analysis of the offered methods errors of parameters definition were introduced and experiments for detecting normalization accuracy dependency from input images type, noise, images physical sizes and quantity of analyzed cuts were carried out. As a practical significance of results it is necessary to specify, that the offered methods have sufficient effectiveness and can be used in technical vision systems, however their application is limited to specific classes of images.

5. REFERENCES

[Rot96] Rothe, I., Suesse, H., Voss, K. The method of normalization to determine invariants. IEEE Trans. on Pattern Analisys and Machine Intelegence 18, No4. pp. 366–377. 1996.

[She99] Shen D., Wong W.-H., Ip H.H.S. Affine-invariant image retrieval by correspondence matching of shapes. Image and Vision Computing 17, pp. 489-499. 1999.

[Pei95] Pei, S.C., Lin, C.N. Image normalization for pattern recognition. Image and Vision Computing 13, No8. pp. 711–723. 1995.

[Cha97] Chang S.H., Cheng F.H., Hsu W.H., Wu G.Z. Fast Algorithm for Point Pattern Matching: Invariant to Translations, Rotations and Scale Changes. Pattern Recognition 30, No2, pp. 311-320. 1997.

[Dau01] Daurat, A. Salient points of Q–convex sets. International Journal of Pattern Recognition and Artificial Intelligence 15, No7. pp. 1023–1030. 2001.

[Mik04] Mikolajczyk K., Schmid C. Scale & affine invariant interest point detector. International Journal of Computer Vision 60, No1. pp. 63–86. 2004.

[Zhu95] Zhuang, X., Zhao, D. A morphological algorithm for detecting dominant points on digital curves. SPIE Proc. «Nonlinear Image Processing VI» 24. pp.372–383. 1995.

[Gol98] Gold, S., Rangarajan, A., Lu, C.P., Pappu, S., Mjolsness, E. New algorithms for 2D and 3D point matching: Pose estimation and correspondence. Pattern Recognition 31, No8. pp. 1019–1031. 1998.

[Spr94] Sprinzak, J., Werman, M. Affine point matching. Pattern Recognition Letters 15, No4. pp. 337–339. 1994.

[Sud97] Sudhir G., Banerjee S., Zisserman A. Finding point correspondence in motion sequences preserving affine structure. Computer Vision and Image Understanding 68, No2, pp. 237-246, 1997.

[Ser82] Serra, J. Image analysis and mathematical morphology. London: Academic Press, 1982.

[Hei94] Heijmans, H.J.A.M. Morphological image operators. Boston: Academic Press, 1994.

[Gia88] Giardina, C.R., Daugherty, E.R. Morphological methods in image and signal processing. New York: Prentice–Hall, Englewood Clifs, 1988.

[Mar02] Maragos P., Schafer, R.W., Butt, M.A., (eds.) Mathematical morphology and its applications to image and signal processing. Computational Imaging and Vision 5. Dordrecht-Boston-London: Kluwer Academic Publishers, 2002.

Combining Multiresolution Shape Descriptors for 3D Model Retrieval

Ryuatrou Ohbuchi
University of Yamanashi
4-3-11 Takeda, Kofu-shi
400-8511, Yamanashi-ken, Japan
ohbuchi@yamanashi.ac.jp

Yushin Hata
University of Yamanashi
4-3-11 Takeda, Kofu-shi
400-8511, Yamanashi-ken, Japan
try@ha.bekkoame.ne.jp

ABSTRACT

In this paper, we propose and evaluate a systematic approach for improving performance of 3D model retrieval by combining multiple shape descriptors. We explored two approaches for generating multiple, mutually independent, shape descriptors; (1) application of a (single-resolution) shape descriptor on a set of multiresolution shape models generated from a query 3D shape model, and (2) application of multiple, heterogeneous shape descriptors on the query 3D shape model. The shape descriptors are integrated via the linear combination of the distance values they produce, using either fixed or adaptive weights. Our experiment showed that both multiresolution and heterogeneous sets of shape descriptors are effective in improving retrieval performance. For example, by using the multiresolution approach, the R-precision of the *SPRH* shape descriptor by Wahl, et al, improved by 8%, from 29% to 37%. A combination of three heterogeneous shape descriptors achieved the R-precision of about 42%; this figure is about 5% better than the R-precision of 38% achieved by the *Light Field Descriptor* by Chen, et al., which is arguably the best single shape descriptor reported to date.

Keywords

3D model database, content-based retrieval, geometric modeling, multiresolution analysis, feature combination.

1. INTRODUCTION

3D shape models are increasingly popular in many application domains, ranging from movie special effects, 3D games on cellular phones or on game consoles, to 3D mechanical CAD/CAE systems. The popularity has prompted research into effective management and reuse of 3D shape models by means of shape-based retrieval of 3D models. An example of such database is the 3D Search engine at the Princeton University [Funkhouser03].

Typical steps for shape similarity based retrieval of 3D models starts with query specification (See Figure 1.) As queries, texts, 2D sketches, 2D images, 3D sketches, and 3D shapes have been used in the past. Multiple query specification methods may be combined, as in the work of Funkhouser et al [Funkhouser03]. The next step is to extract feature, or *shape descriptor* (SD), from the query. Also, as a pre-computing step, SDs for 3D shape models in the database have been computed. The querying method and the 3D shape representation used for the database influences the shape descriptor and the similarity (or more often, distance) computation method. During the distance computation, it is often desirable to reflect human judgment. The database retrieves and presents the models most similar to the query based on the distances computed.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

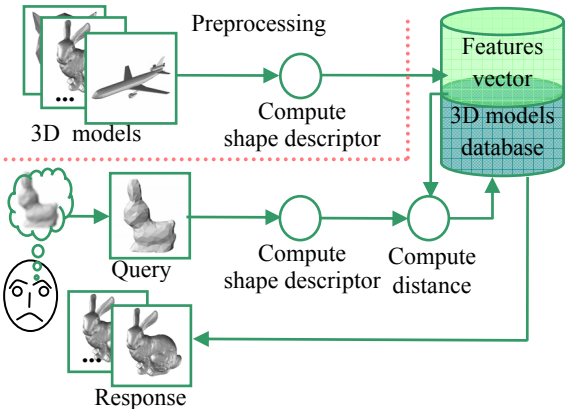


Figure 1. A typical 3D model database system.

In this paper, we explore an approach to boost shape similarity retrieval performance of a 3D model database *by extracting more features* from shape models. Our method uses two method to try to extract more shape feature from a query 3D model; (1) extraction of a multiresolution set of features by applying a SD to the models having multiple resolution levels generated by using the method by Ohbuchi, et al [Ohbuchi03], and (2) extraction of a heterogeneous set of features by applying multiple, heterogeneous SDs to the model.

Multiple SDs are integrated via distance value they produce by using linear combination, which allows integration of features that does not explicitly produce feature vectors, namely, the *Light Field Descriptor* (*LFD*) [Chen03] by Chen, et al. Integration of a set of distances are done through either *fixed weight* linear combination of distances, or knowledge based *adaptive weight* linear combination of distances. The latter is a modification of the "purity" based method of Bustos, et al. [Bustos04a, Bustos04b].

Our experiments showed that the multiresolution approach boosted performances of many, but not all, of the SDs we tested. Combinations of multiresolution, heterogeneous shape descriptors produced the best performance of the combinations we tested. For example, one of the combinations produced R-precision of 42%, which is about 5% higher than the 38% achieved by the LFD by Chen, et al., which is one of the best performing SDs according to Tangelder, et al. [Tangelder04].

This paper is organized as follows. In the following section, we review the previous integrate features for 3D model retrieval. In Section 3, we describe the method to compute multiresolution SDs, and the method to integrate multiple SDs by their distance values. A set of performance evaluation experiments and their results are described in Section 4. We conclude the paper in Section 5.

2. Previous Work

In the field of content based search and retrieval of 2D images, it is typical to extract more than one feature from an image, and combine these features for an overall similarity comparison. Thus it is natural to think of such an approach for the 3D model retrieval. This approach has been taken by several groups; Iyer, et al. [Iyer03], Bustos et al [Bustos04a, Bustos04b], and Atmosukarto, et al. [Atmosukarto05].

Iyer et al. weighted and combined multiple heterogeneous feature vectors, letting the user control the weights explicitly through a user interface or implicitly through a relevance feedback mechanism that employs interactive learning. The method by Atmosukarto et al. also weighted and combined heterogeneous feature vectors. Their experiments showed that combinations of descriptors have better performance than any single shape descriptor they evaluated.

Unlike the former two, the method by Bustos et al. integrated multiple, heterogeneous features via *distance*. Bustos computed the overall distance between a pair of models as a linear combination of the distances using either fixed or adaptive weights. In the fixed weight combination, weights are the same regardless of the model (or the model category.) In the adaptive weights combination, weights are computed by using *purity*, which is an estimate of the performance of the SD determined by using a pre-classified training database. Bustos et al. reported significant performance gain using both fixed weight and adaptive weight linear combination of distances, although the adaptive one performed better.

Integration of multiple shape descriptors can be achieved using either (1) *feature vectors* of the descriptors, if available (the approach by Iyer et al. and Atmosukarto, et al.), or (2) *distance values* computed using the descriptors (the approach by Bustos et al.). Integration using feature vectors potentially allows fine tuning of distance computation, e.g., by weighting each element of the vectors. However, this approach can't be used if a shape descriptor produces distance but not feature vector. For example, one of the most powerful 3D shape descriptors, the LFD by Chen et al. [Chen03] produces distance only, and is useful only for distance based integration. Findings by Atmosukarto et al. and by Bustos et al. are contradictory regarding whether the shape descriptor should be combined as feature vectors or as distances. Atmosukarto reported that a combination of distances does not produce any performance improvement. Bustos et al., on the other hand, reported that combinations of distances are beneficial. Our finding reported in this paper agrees with that of Bustos et al.

3. METHOD

In this section, we describe the method to compute multiresolution shape descriptors and the method to combine multiple shape descriptors by their distance values.

3.1. Computing multiple shape descriptors

To improve retrieval performance through feature integration, features should capture as different shape feature of the model as possible. Our method uses the

following two different approaches to extract mutually independent shape descriptors.

1. **Multiresolution approach**: Generate a set of multiresolution (MR) shape models from a model to be compared. A (single-resolution) shape descriptor is applied to the MR shape models to produce multiple SDs.
2. **Heterogeneous shape descriptor approach**: Apply multiple shape descriptors to the model to be compared.

A combination of the two approaches above is also possible. For example, an MR representation having m levels may be combined with n heterogeneous shape descriptors to produce $m \times n$ shape descriptors.

3.1.1 Multiresolution shape descriptors

The method by Ohbuchi, et al. [Ohbuchi03] compares 3D models at multiple scales following the steps below:

1. **Compute a multiresolution representation**: The surface-based input model is converted into a point-based model by *Monte-Carlo* sampling of the surfaces of the model. A set of L-1 scale values α_i, i=1…L-1 is computed based on the size of the model. The set of scale values are used to normalize size among shape models. Then, compute L-1 3D alpha shapes from the point set model by using the L-1 scale values α_i. Of L shape features, those of the coarser (L-1) levels are computed by using the 3D alpha shapes [Edelsbrunner94]. For the finest resolution level L, however, the original, polygon soup model is used. Ohbuchi et al calls this set of multiresolution models *Alpha-Multiresolution Representation* (AMR).
2. **Compute multiresolution shape descriptors**: Apply a shape descriptor x to a model at each resolution level of the AMR, creating a set of multiresolution shape descriptors AMR-x.
3. **Compute distance between a pair of features**: Compute a distance between AMR-x shape descriptors of a pair of models to be compared, using a statically weighted linear combination of L distances.

An advantage of the AMR above is that it can be computed for polygon soup models or even for point set models (without the step 1 above.) Figure 3 shows an example of AMR representation for a surface-based 3D model of rabbit. The AMR is reminiscent of morphological multiresolution representations for 2D images.

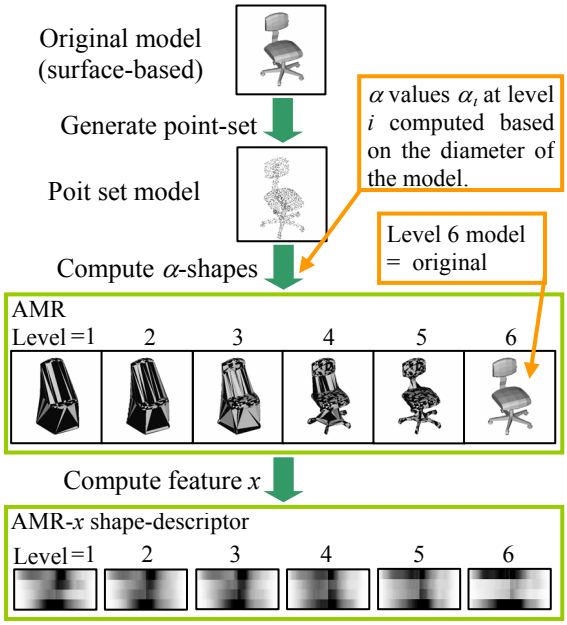


Figure 2. Computing the AMR-x multiresolution shape feature.

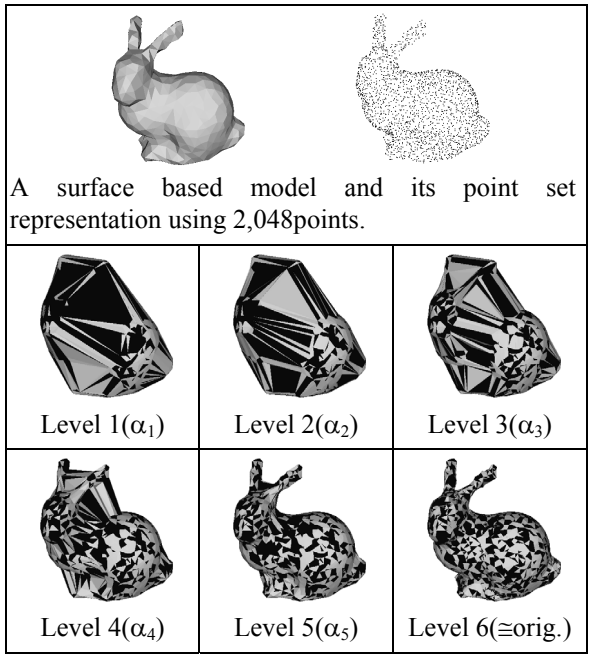


Figure 3. An example of AMR representation for the surface-based model of a rabbit.

3.1.2. Heterogeneous shape descriptors

As the heterogeneous shape descriptors, we used the *D2* by Osada, et al.[Osada02], the *AAD* by Ohbuchi et al. [Ohbuchi05], the *SPRH* by Wahl, et al. [Wahl03], and the *LFD* by Chen, et al. [Chen03]. The D2 is a 1-dimensional (1D) histogram of distances between every pair of points generated on surfaces of a model. The AAD and the SPRH are both extensions to the D2 above. In addition to the

distance used in the D2, the AAD and the SPRH extract such features as the mutual orientation of the pair of points, resulting in a 2D histogram for the AAD and a 4D histogram for the SPRH. The LFD by Chen, et al. is different from all of the above. It compares similarities of a set of images generated from multiple viewpoints about the 3D model. According to the survey paper by Tangelder et al. [Tangelder04], the LFD is by far the most powerful shape descriptor.

3.2 Integrating multiple shape descriptors

In combining distances produced by shape descriptors, our method normalizes the distances first, and then computes a weighted linear combination of the distances.

3.2.1 Normalization

Prior to integrating distances, the distances are normalized by their standard deviations. Let U be the set of 3D models in the database, and $o \in U$ be the model from the database, and SD_i, $1 \le i \le N$ be the shape descriptors. For the shape descriptor SD_i, rhe average $\mu(d_i)$ and the standard deviation $\sigma(d_i)$ are computed for the database U. Let $d_i(q,o)$ be the distance prior to normalization computed using the SD_i for the model pair q and o. Then the normalized distance $\hat{d}_i(q,o)$ for the SD_i is computed as below.

$$\hat{d}_i(q,o) = \frac{1}{2}\left(\frac{d_i(q,o) - \mu(d_i)}{3 \cdot \sigma(d_i)} + 1\right) \qquad (1)$$

Figure 4 shows examples of histograms of distances for the four SDs we have used. The distance axis is normalized to its maximum distance value (=100%). It can be seen that the normalization using standard deviation would perform better than the normalization using maximum distance.

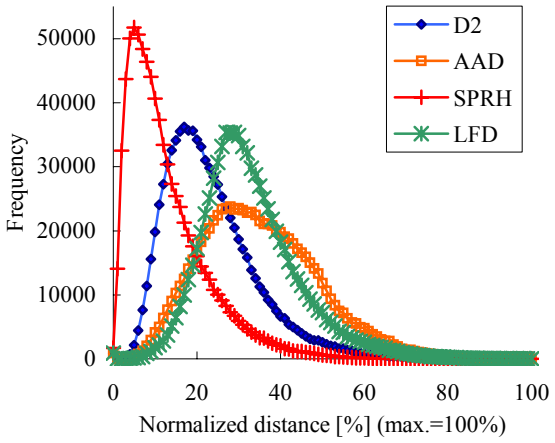


Figure 4. Distribution of distances for the SDs.

Integrating shape descriptors using normalized distance using the formula (1) above should perform better than our previous method described in [Ohbuchi03], which employed no distance normalization at all in integrating SDs generated by using the AMR. Also, we expect our normalization method using standard deviation of distances to be more robust against outliers than the normalization method using maximum distance employed by Bustos, et al. [Bustos04a].

3.2.2. Weighted Linear Combination

Our method computes the overall distance $d(q,o)$ as the linear combination of N normalized distances $\hat{d}_i(q,o)$ using the following formula.

$$d(q,o) = \sum_{i=1}^{N} w_i \hat{d}_i(q,o) \qquad (2)$$

We compared two different methods to determine the weights w_i.

1. Fixed weighting: weights $w_i^{fix.}$ are predetermined and fixed across the query.
2. Adaptive weighting: weights $w_i^{adap.}$ are adaptively varied according to the query and its (estimated) category.

Using fixed weights has two drawbacks. One is that the fixed weights won't produce the best performance across all the classes. Assuming a pre-categorized database, the performance of a SD varies depending on the class the query model (is supposed to) belongs to. That is, for example, one SD is good at querying human figures while another SD is good at querying airplanes. The other is that experimentally finding a best set of weights for a given set of SDs and a database can be computationally expensive for a nontrivial number of SDs and models. An adaptive weighting method that adapts to the query model and/or to the class in which the query model is likely to belong is quite important.

For the adaptive weighting, we adopted Bustos's "*purity*" based weighting scheme [Bustos04a, Bustos04b] with a few minor modifications. The purity is an estimate of "goodness" of a SD in characterizing a category in a given database. The idea behind the purity is the maximum information gain criteria for selecting an attribute in splitting a set in the decision tree learning algorithm. The purity assumes that the database of the 3D models is pre-classified into M classes. Let S_i^k be the number of models retrieved from a class C_k, $1 \le k \le M$, using the shape descriptor SD_i. The purity $purity(SD_i, q, t)$ is computed as below for a shape descriptor SD_i, query q, and a positive integer constant t.

$$purity(SD_i, q, t) = \max_{1 \le k \le M} \left(\left| S_i^k \right| \right) \quad (3)$$

In other words, the purity for the SD_i is higher if the SD returned more model from the category C_k in the top t retrievals, regardless of the class.

Using the $purity(SD_i, q, t)$, the adaptive weight $w_i^{adap.}$ between the query q and the 3D model $o \in U$ is computed as follows.

$$w_i^{adap.} = purity(SD_i, q, t)^n \quad (4)$$

Bustos, et al. used a slightly different $w_i^{adap.}$. Our version could have a larger difference in weights, depending on the selection of the "purity power" parameter n .

$$w_i^{adap.} = purity(SD_i, q, t) - 1 \quad (5)$$

In the following, we call our weighting method purity* and Bustos's original adaptive weighting method purity.

4. EXPERIMENTS AND RESULTS

We implemented our own versions of the D2 [Osada02] and the SPRH [Wahl03] shape descriptors. We used our original implementation of the AAD [Ohbuchi05] shape descriptor. For the LFD [Chan03], we used the executable provided by the original authors of the LFD paper [Chan03] found at their web site. In our previous work [Ohbuchi03], we combined the AMR only with the AAD shape descriptor. In this work, we combine the AMR approach with all the four shape descriptors listed above.

For the performance evaluation experiments, we used the *Princeton Shape Benchmark* (PSB) [Funkhouser03] database. It contains the total of 1914 models, divided into the 907 model training set and the 907 model test set. Each set is classified into 90+ "base" classes. We used the most detailed "base" classifications for the experiment.

As quantitative measures of performance, we used the *R-precision* (1R), the *2R-precision* (2R), the *11 point average precision* (11P) figures, and the precision-recall plot [Baeza-Yates99]. The R-precision is the ratio, in percentile, of the models retrieved from the desired class C_k (i.e., the same class as the query) in the top R retrievals, in which R is the size of the class $|C_k|$. The 2R-precision is similar to the R-precision, except that the figure is computed using the top 2R retrievals. In computing the R- and 2R-precision values, the query q is not counted as the retrieved model. That is, the q is drawn from the database U , the 1R and 2R values are computed by using $|C_k - 1|$. The 11-point average 11P is the average of precision values taken at 11 equally spaced recall values {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}. A 11P average precision value can be considered as a summary of the recall-precision plot, which emphasizes overall performance. The 1R and the 2R values favor methods having higher precision for the "near the top" retrievals.

4.1. Multiresolution shape descriptors

In this experiment, we evaluated the effectiveness of integrating multiresolution shape descriptors. We first compared the retrieval performances of the shape descriptors at different resolution levels. Interestingly enough, as shown in Table 1, the most detailed models (the original models) may not achieve the highest retrieval scores. In the cases of the D2 (not shown), the AAD, and the SPRH, retrieval using coarser scales (the level 5, and to a lesser degree, the level 4), produced better 1R score than using level 6 (i.e., most detailed) models. For the LFD, however, retrieval using the most detailed models produced the highest 1R score. A possible explanation for this is that the LFD favors shape details of the models and that the corners and edges of the convex hull models interfere with the performance of the shape descriptor. Note also that retrieval using the coarsest level (i.e., convex hull) models performed surprisingly well.

As shown in Table2 and in Figure 5, for some of the SDs, combinations of multiresolution (MR) shape descriptors using either the fixed or the adaptive weights significantly outperformed their single-resolution (SR) counterparts. In the case of the AAD, 1R figure improved by 10% from 24.4% for the SR version to 34.9% for the MR version using the purity* weighting. The performance of the SPRH increased from 28.6% (SR) to 36.6% (MR), approaching the 38.0% of the LFD. The performance of the LFD, however, did not improve due to the MR combination. Also, adaptive weighting using the purity* showed small but consistent advantage over their fixed weighting counterparts for all of the four SDs tested.

Table 1. R-Precision varies across resolution levels. As the numbers show, the most detailed model may not be the best choice for retrieval.

Resolution levels	R-precision [%]			
	D2	AAD	SPRH	LFD
1	18.53	21.50	24.50	28.32
2	18.12	21.97	25.67	28.71
3	19.96	24.02	28.99	30.62
4	20.27	25.54	30.42	32.59
5	20.92	26.78	31.15	34.83
Orig=6	18.72	24.41	28.57	37.96

Table 2. The multiresolution (MR) versions of D2, AAD, and SPRH outperform their single resolution (SR) counterparts. Such is not the case for the LFD.

SDs	Weights	1R	2R	11P
D2		18.72	27.23	0.314
MRD2	Fixed, 111111	25.14	38.84	0.375
	Fixed, 123456	24.62	33.34	0.369
	Adaptive, purity*	25.45	33.54	0.379
AAD		24.41	34.40	0.364
MRAAD	Fixed, 111111	35.11	44.81	0.446
	Fixed, 123456	33.19	43.06	0.456
	Adaptive, purity*	34.89	45.38	0.471
SPRH		28.57	38.66	0.404
MRSPRH	Fixed, 111111	35.11	44.81	0.468
	Fixed, 123456	35.27	45.69	0.470
	Adaptive, purity*	36.60	45.08	0.482
LFD		37.96	48.74	0.496
MRLFD	Fixed, 111111	35.90	44.22	0.474
	Fixed, 123456	37.10	45.43	0.484
	Adaptive, purity*	37.39	45.61	0.489

4.2 Weighting methods

We compared the performance of the purity* based weighting for different values of the parameter *purity power n* in the equation (2). We also compared the performance of our purity* and the original purity by Bustos, et al. [Bustos04a].

Table 3 shows the performance of the MRLFD-MRAAD-MRSPRH combination using different values of *purity power n*. In terms of 1R scores, $n = 3.0$ produced the best 1R performance, and $n = 9$ produced the best 11P performance. Table 4 compares the Bustos's purity with our purity* for their retrieval performance. In all the combinations tested, our purity* using the selected parameter n performed better than the original purity of Bustos et al. Our purity* performed better probably because the weight can have a larger dynamic range. As a disadvantage, our scheme requires a search for the bester value n, although the search should be relatively easy for it is a 1D search space.

Table 3. Effects of purity power *n* on the retrieval performance for the MRLFD-MRAAD-MRSPRH combination.

	N	1R	2R	11A
Bustos's purity		42.01	52.02	0.538
Purity*	1.0	41.26	51.49	0.527
Purity*	3.0	42.40	52.34	0.550
Purity*	5.0	42.30	51.97	0.562
Purity*	9.0	41.53	50.86	0.567
Purity*	30.0	40.33	49.30	0.563

Table 4. Comparison of retrieval performance between Bustos's *purity* v.s. our *purity**.

Descriptors	Weights		1R	2R	11P
MRAAD	purity		33.19	42.77	0.455
	purity*	*n*=2.0	33.19	43.06	0.456
MRSPRH	purity		35.64	44.75	0.451
	purity*	*n*=3.0	36.60	45.07	0.482
SPRH-AAD	purity		27.69	37.05	0.385
	purity*	*n*=4.0	29.61	39.54	0.417
LFD-AAD	purity		39.15	49.11	0.497
	purity*	*n*=2.0	41.60	51.93	0.531
LFD-SPRH	purity		40.95	50.55	0.517
	purity*	*n*=2.0	42.46	52.68	0.539
LFD-SPRH-AAD	purity		40.57	50.72	0.518
	purity*	*n*=3.0	42.72	52.70	0.542

4.3 Combination of heterogeneous shape descriptors

In this experiment, we compared the performance of integrated SDs using both multiresolution and heterogeneous combinations of various SDs. Table 5 summarizes the experiment. In all the results listed, we used the purity* adaptive weightings. In each combination, a best performing parameter *n* is chosen out of the 10 candidates values of *n*={1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0}. The Figure 6 shows the recall-precision plot of five combinations selected from the Table 5.

Table 5 and Figure 6 clearly show that combining multiple, heterogeneous shape descriptors via distances could produce significant performance gain compared to any one of the single resolution SDs. As a reference, the LFD has the 1R precision of 38%. The combination of LFD, AAD, and SPRH produced nearly 5% performance gain over that of the LFD, resulting in the 1R precision of 42.5%.

Table 5. Performance comparison of some of the combinations tested using both heterogeneous and multiresolution shape descriptors. All the combinations used the purity* adaptive weighting.

Shape descriptors	1R	2R	11P
LFD	37.96	48.74	0.496
SPRH	28.57	38.66	0.404
MRSPRH	36.60	45.08	0.482
SPRH+AAD	29.61	39.54	0.417
MRSPRH+MRAAD	37.02	46.96	0.491
LFD+AAD	41.60	51.93	0.531
LFD+MRAAD	39.81	49.94	0.520
LFD+SPRH	42.52	52.68	0.537
LFD+MRSPRH	42.71	51.74	0.541
LFD+AAD+SPRH	42.72	52.70	0.542
LFD+MRAAD+MRSPRH	41.37	51.15	0.534
MRLFD+MRAAD+MRSPRH	42.31	51.97	0.543

On the other hand, combinations of the LFD with the MR shape descriptors did not produce consistent and significant performance gain. The reason for this may be attributed to the AMR model, the purity* adaptive weighting scheme, or both. Further study is needed to determine the exact cause.

5. CONCLUSION AND FUTURE WORK

In this paper, we explored a systematic approach for improving shape-based retrieval performance of 3D shape models. Our approach is to (1) extract as many (mutually independent) shape features as possible, and (2) combine the distances computed using these features by using an adaptive weighting scheme. To extract different shape features, the method employed a combination of multiresolution shape descriptors and heterogeneous shape descriptors. We used the 3D alpha-shape [Edelsbrunner94] based method we have proposed previously [Ohbuchi03] to capture multiresolution shape features. To combine distances computed using these shape descriptors, we adopted Bustos's purity weighting scheme with slight modification.

Experiments showed that the proposed method of integrating multiresolution and heterogeneous shape descriptors is effective in improving retrieval performance. Many combinations of the shape descriptors we tested surpassed the performance of the arguably the best (single) shape descriptor, the Light Field Descriptor by Chen, et al. [Chen03].

We intend to explore better adaptive weighting schemes and better multiresolution shape feature extraction approaches.

6. ACKNOWLEDGMENTS

This research is funded, in part, by the Grants-in-Aid for Scientific Research from the *Japan Society for the Promotion of Science* (No. 17500066).

7. REFERENCES

[Atmosukarto05] I. Atmosukarto, W.K. Leow, Z. Huang, Feature Combination and Relevance Feedback for 3D Model Retrieval, *Proc. 3DPVT* 2005, pp. 334-339, (2005).

[Bustos04a] B. Bustos, D. Keim, D. Saupe, T. Schreck, D. Vranić, Automatic Selection and Combination of Descriptors for Effective 3D Similarity Search, *Proc. IEEE MCBAR'04*, pp. 514-521, (2004).

[Bustos04b] B. Bustos, D. Keim, D. Saupe, T. Schreck, D. Vranić, Using Entropy Impurity for Improved 3D Object Similarity Search, *Proc. IEEE ICME 2004* (2004).

[Chen03] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, M. Ouhyoung, On Visual Similarity Based 3D Model Retrieval, *Computer Graphics Forum*, **22**(3), pp. 223-232, (2003).

[Edelsbrunner94] H. Edelsbrunner, E. P. Mücke, Three-dimensional Alpha Shapes, *ACM TOG*, **13**(1), pp. 43-72, (1994).

[Funkhouser03] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, D. Jacobs, A search engine for 3D models, *ACM TOG*, **22**(1), pp. 83-105, (2003).

[Funkhouser04] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, David Dobkin, Modeling by ex-ample, *ACM TOG*, **23**(3), pp.652-663 (2004).

[Iyer03] Iyer, N., Kalyanaraman, Y., Lou, K., Jayanti, S., Ramani, K., A Reconfigurable, Intelligent 3D Engineering Shape Search System Part I: Shape Representation, *Proc. ASME DETC* '03, 23rd CIE Conf. (2003).

[Iyer05] M. Iyer, S. Jayanti, K. Lou, Three Dimensional Shape Searching: State-of-the-art Review and Future Trends, *Computer Aided Design*, **5**(15), pp. 509-530, 2005.

[Leifman03] G. Leifman, S. Katz, A. Tal, R. Meir., Signatures of 3D Models for Retrieval, *The 4th Israel-Korea Bi-National Conf. on Geom. Modeling and Comp. Graph*., February 2003, 159-163. (2003).

[Ohbuchi05] Ryutarou Ohbuchi, Takahiro Minamitani, Tsuyoshi Takei, Shape-similarity search of 3D models by using enhanced shape functions, *International Journal of Computer Applications in Technology* (*IJCAT*), **23**(3/4/5), pp. 70-85, (2005).

[Ohbuchi03] R. Ohbuchi, T. Takei, Shape-Similarity Comparison of 3D Shapes Using Alpha Shapes, *Proc. PG 2003*, pp. 293-302, IEEE Press, (2003).

[Osada02] R. Osada, T. Funkhouser, B. Chazelle, D. Dobkin, Shape Distributions, *ACM TOG*, **21**(4), pp. 807-832, (2002).

[Shilane04] P. Shilane, P. Min, M. Kazhdan, T. Funkhouser, The Princeton Shape Benchmark, *Proc. SMI '04*, pp. 167-178, (2004).
http://shape.cs.princeton.edu/search.html

[Tangelder04] J. Tangelder, R. C. Veltkamp, A Survey of Content Based 3D Shape Retrieval Methods, *Proc. SMI '04*, pp. 145-156.

[Wahl03] E. Wahl, U. Hillenbrand, G. Hirzinger, Surflet-Pair-Relation Histograms: A Statistical 3D-Shape Representation for Rapid Classification, *Proc. 3DIM 2003*, pp. 474-481, IEEE Press, (2003).

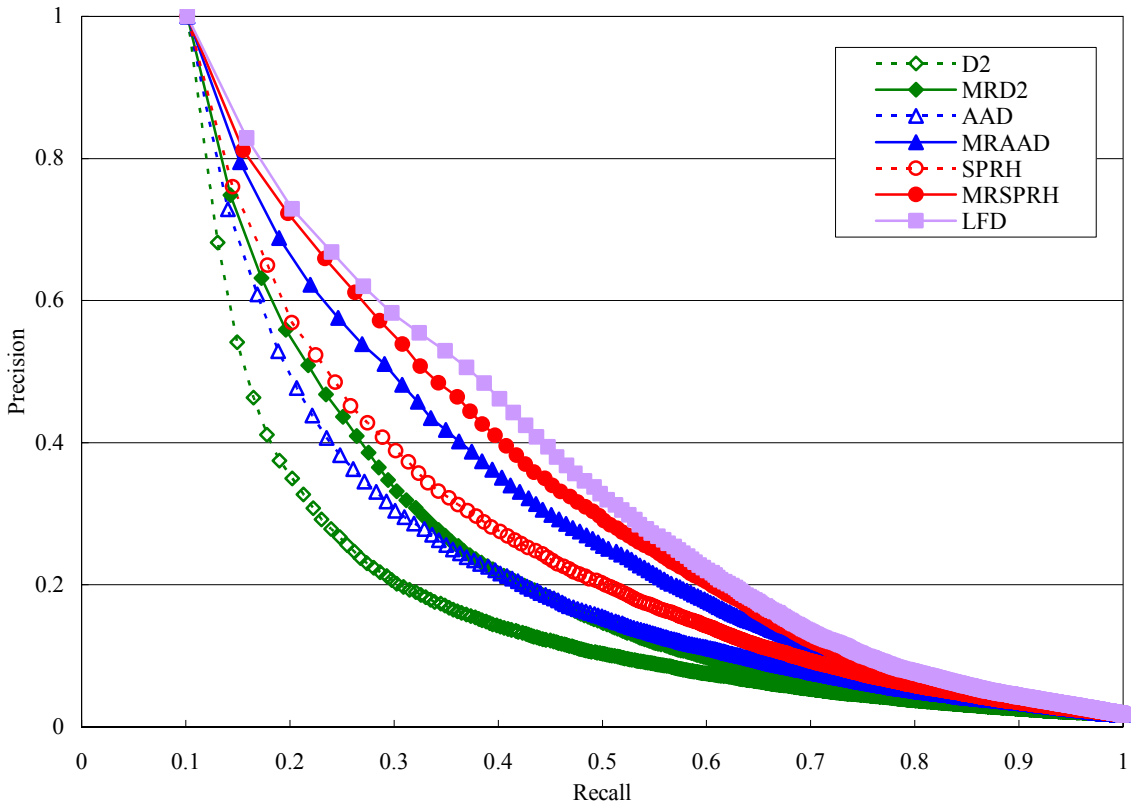


Figure 5. Performance gain due to the multiresolution approach. The multiresolution shape descriptors using AMR increased performance of many but not all of the SDs.

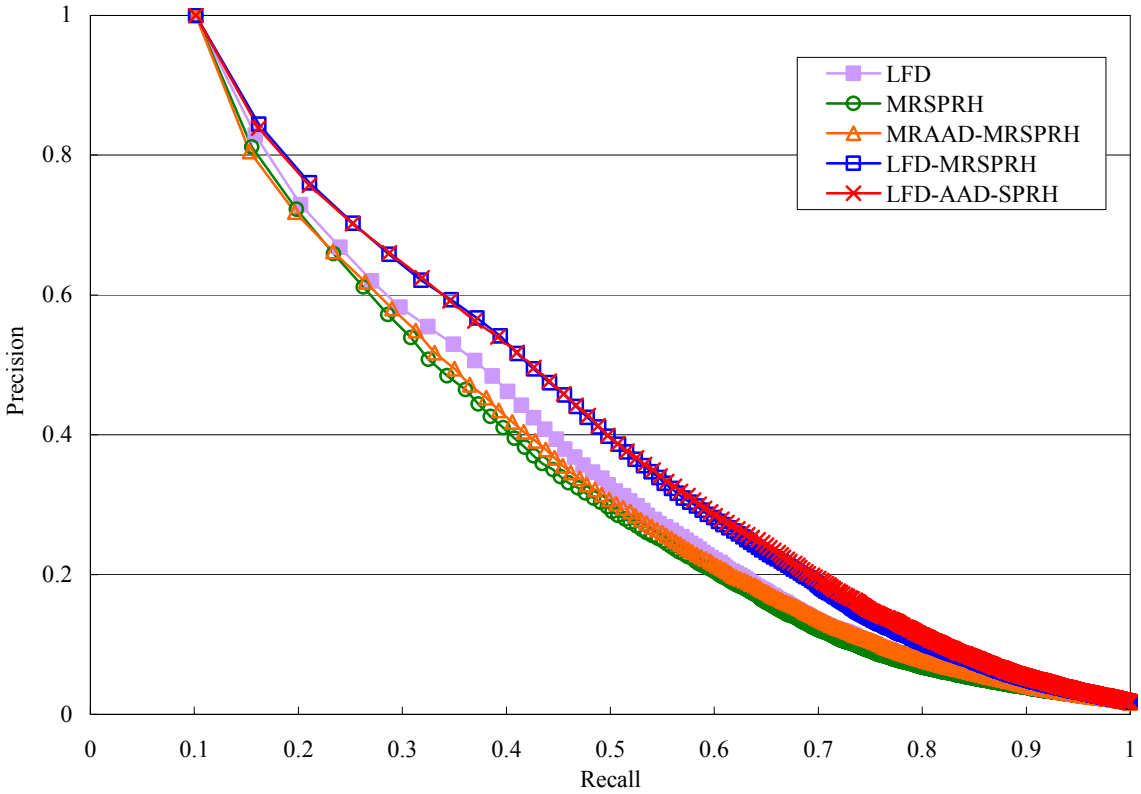


Figure 6. Precision-recall plots of some of the combinations of descriptors. Some of the combinations significantly outperform the LFD [Chen03], arguably the best single shape descriptor to date.

Scanner morphing simulation with image warping

Jordi Santonja Blanes	Jordi Linares Pellicer	David Cuesta Frau	Pau Micó Tormos
D. Inform. Sist. i Comp.	D.Sist. Inf. i Computació	D. Inform. Sist. i Comp.	D. Inform. Sist. i Comp.
U. Politècnica València	U. Politècnica València	U. Politècnica València	U. Politècnica València
EPSA. Plaça Ferrandis, 2	EPSA. Plaça Ferrandis, 2	EPSA. Plaça Ferrandis, 2	EPSA. Plaça Ferrandis, 2
Spain 03801 Alcoi	Spain 03801 Alcoi	Spain 03801 Alcoi	Spain 03801 Alcoi
jorsanbl@disca.upv.es	jlinares@dsic.upv.es	dcuesta@disca.upv.es	pabmitor@disca.upv.es

ABSTRACT

In this work, a new image deformation method based on a copy-art technique is described. This copy-art process involves the use of either a photocopier or a linear scanner with a motif in motion as the light bar sweeps by to create distortions. The new method provides an easy-to-use tool which is very intuitive and offers predictable results. This allows the artist to recreate the original copy-art process digitally. In order to define the transformation completely, the user has to specify three elements for the original image interactively: path, rotation and velocity. To get the final image, two methods can be followed: either to sample each line from the original image to the result image using forward mapping or to split the transformation into a grid. This latter approach will allow more control in the final result.

Keywords

Warping, morphing, scanner, deformation, images.

1. INTRODUCTION

Electrography or copy-art is a set of analog techniques involving the interaction of light and electricity to produce or transform images. Usually the device is a photocopier, and the process can be, for example, to make multiple-generation copies to obtain a dissolved image from the original. Another copy-art technique consists of moving the original image or object over the photocopier as the light bar sweeps by to create distortions from the original [Bern].

This analog distorting technique allows the designer or artist to create a large variety of distortions with a high level of expressivity. The artist can change the velocity of the original image while it is scanned (see Fig. 1) or can translate or rotate the image freely to obtain any distortion.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

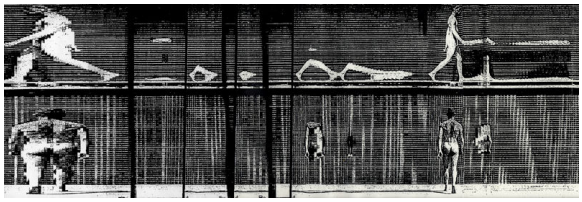


Figure 1. Work by J. Llopis and Rebeca Padin. 1992 Mide. Cuenca. [MIDE]

In this paper a digital equivalent method is presented. This method consists of deforming the original image following a line and some properties of it; all traced by the user. This approach offers some advantages: an easy-to-use interface, highly predictable results, the modification of the mesh after its generation and the possibility to repeat the complete process to apply the same transformation to a new input image. All these characteristics offer a new artistic tool to designers.

2. PREVIOUS WORK

Some methods have already been described to warp an image. These methods can be classified according to the type of transformation [Gom99]:

- Parameter-Based Techniques. This class of techniques, more than deforming an image or an object, is a modelling technique. A set of parameters, such as scale, twisting and blending, really defines the shape of an object. The modification of these parameters

leads to a deformation of the original image or object [Barr84].

- Feature-Based Techniques. The user must provide a finite set of geometric features on the source image, and their corresponding counterparts on the target image. The deformation is defined by binding all the source and target features. A clear example is the two-dimensional field-based technique [Bei92].

- Free-Form Techniques. This group of techniques deals with a specification of the deformation based on coordinate systems, and using free-form curves (B-Splines, Bézier, etc.) to define the coordinate systems. The user must change the control points of the coordinate curves to define a warping [Ara95][Ara98]. The FFD (Free Form Deformation) has in [Seu96] a very good result.

- Hybrid Techniques: Some authors have combined the advantages of the previous techniques and have even developed a complete deformation environment to offer more flexibility [Mil02].

Scanner deformation

Regarding the deformation done by a linear scanner, some papers that can be found in the bibliography analyze this process in order to rectify it and obtain the original image [Wol95][Pil02]. The techniques used to unwarp the distorted image are based on an analysis process over the target, looking for known features from the source image.

In this paper a new method to warp an image is described. This method simulates the deformation obtained when an image is moved while it is being captured with a linear scanner. This method must follow these requirements:

- It must allow foldover (many-to-one mapping) of the source image. A section of the source image could pass twice or more times over the light beam when the user moves it.

- Highly predictable. The user must predict the resulting image with the deformation parameters provided.

- Fast. The definition and calculation of the deformation must be fast and easy for the user.

- Intuitive. The deformation method is based on a real analog distortion technique, so its definition must resemble the real process.

3. PROCESS DESCRIPTION

The process to be emulated consists of two elements: the capturing device and the input image.

The capturing device gets the input image by sweeping a light bar along its capturing area. It acquires the image line by line as the bar moves at a constant speed over the area. See Fig. 2.

The image to acquire is placed in the capturing area, usually remaining without movement. If the user needs, as a result, a distortion of this image, this image can simply be moved on this area as the light bar sweeps by (see Fig. 3). Depending on the movement, the user can obtain different results. For example, the image in Fig. 1 was done moving the original image parallel to the direction of movement of the light bar, allowing it to pass more than once over any portion of the original image. If the user rotates the original image, a completely warped image will be obtained.

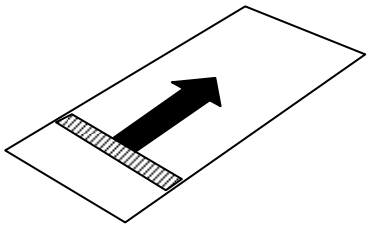


Figure 2. Capturing device: displacement of the light bar.

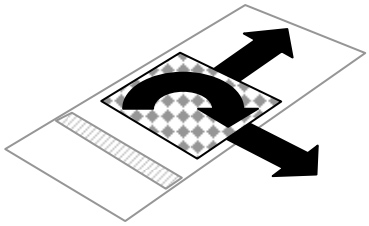


Figure 3. Original image movement as the light bar sweeps by.

4. DEFORMATION

Image warping

The transformation between the source and the final images is represented in Fig. 4. This image represents a transformation from the source space (the original image) to the target space (the final image) [Wol90]. As the target space is directly obtained as the light bar moves uniformly capturing lines from the source space, the final space is

represented as a regular undistorted and orthogonal grid.

The source space represents the area of the original image to be mapped into the target space. As the original image can be moved and rotated, the original space must represent these distortions, hence the distortion of the original space. To obtain the final image from the original, the source space must be mapped on the target space. Note that with this definition, the source space can fold over itself.

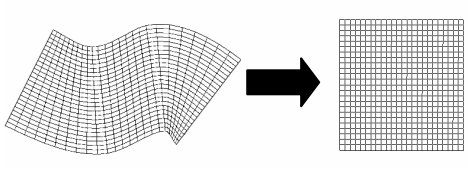


Figure 4. Scheme of the deformation system: original and target spaces.

Deformation definition

In order to define the deformation, the user must provide the movement of the original image to define the source space. This movement has been divided into three items: path, rotation and speed. These three items completely define the transformation that a user can create on a scanner.

4.1.1.1 Path definition

The first step involves the creation of a path that the light bar will follow over the image when it moves. In order not to distort the image, this path must be completely straight. To create a deformation, the user must create a curve by tracing some control points. These control points define a natural spline, and are pass points of the curve. This approximation has been chosen because of its simplicity of use.

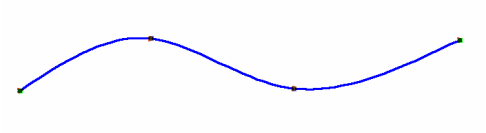


Figure 5. Path definition.

4.1.1.2 Rotation definition

Once the path has been traced, the preliminary rotations are calculated automatically as perpendicular to the trajectory (see Fig. 6-up). These rotations define the rotation of the light bar as it goes over the original image. If the rotation lines are all parallel the image is simply moved without rotation.

After the path is traced and the preliminary rotations are calculated, the user can freely define the rotation on every control point, simply by clicking and dragging the lines on every point (see Fig. 6-down).

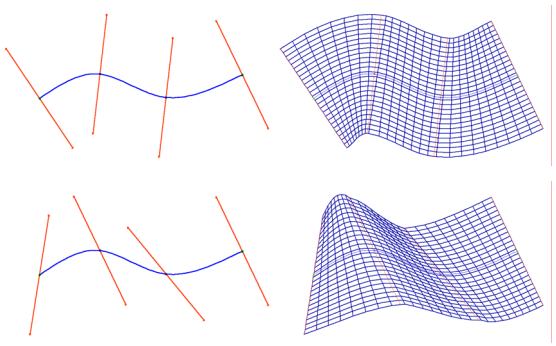


Figure 6. Rotation modification: preliminary rotations perpendicular to the path, and after user modification.

4.1.1.3 Speed definition

The third modification the user can define is the velocity, that defines the speed of the light bar over the original image. Defining a constant velocity makes no contractions or elongations of the central part of the image when is warped (the sides can change in size if the path traces a curve).

If the user defines different speeds along the path, the final image will show contractions or elongations, as shows Fig. 1.

To define the speed, the user must edit a bar diagram, where the vertical component is the speed of one segment in the path, and all the control points of the path are placed in the horizontal component. If the user rises one of the lines, he/she increases the speed of the corresponding segment between control points (see Fig. 7)

Figure 7. Speed modification: central segment has a higher velocity.

The user can see the result of the change in the mesh that represents the final transformation, as shown in Fig. 8. See below a description of how this mesh is obtained.

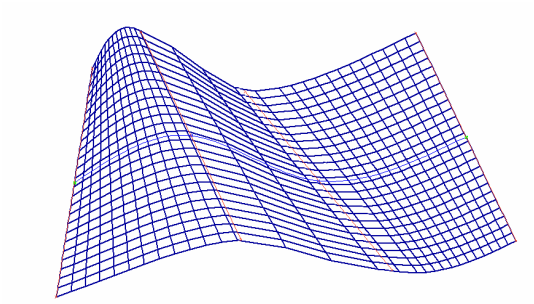


Figure 8. Mesh result of the speed modification.

5. MESH GENERATION

When the deformation parameters are completely defined, or during its definition, the user can see how the distortion will be with a deformation mesh that follows all the defined parameters: path, rotations and velocities. Through this mesh the user can foresee the final result image, because the mesh is constructed with the original space deformation.

As the final image is a rectangle, it is easy to divide this rectangle into a grid with constant spaces between rows and columns. If the source space is defined using a mesh with the same number of rows and columns, the final transformation is simply to apply every quadrilateral of the source mesh to every rectangle in the target grid (see Fig. 4).

To create this grid a two step process takes place: first, to calculate the subdivision of the path into the number of required parts following the speed definition, and second, to interpolate the rotations between parts.

Path subdivision

The calculation of the subdivisions is trivial when the velocity is constant along the path, every segment is simply the subdivision of the path length into the number of required parts (see Fig. 9a).

If there are velocity variations along the path, the parts with higher velocity must have fewer and larger segments, whereas the low velocity parts must have more and shorter segments.

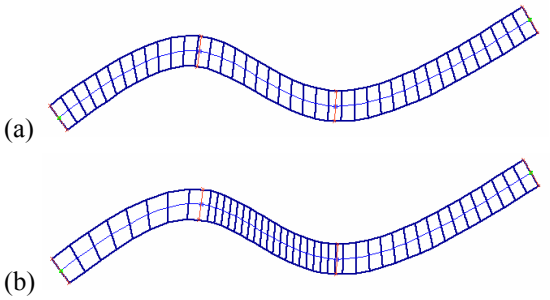


Figure 9. Constant speed path subdivision (a) and variable speed (b).

In order to obtain this result, the subdivision of the total length into the segments is done using the 'virtual length' of the path, not the real length, and the resulting segments are repositioned on the real length of the path. This virtual length is the result of multiplying the real length by the velocity relation. See Fig. 10.

In the final mesh, as every source quadrilateral is mapped on a quadrilateral of the target grid, the more speed it has the bigger source quadrilaterals are, obtaining a final image with a smaller sampled area as the scan of that area takes less time. See Fig. 1 and Fig. 10.

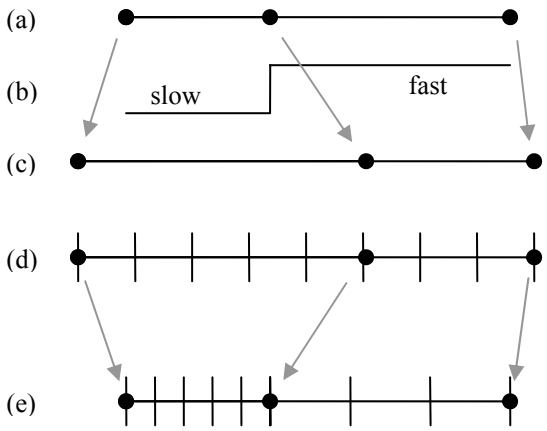


Figure 10. Calculation of the segments along the path: (a) real length path, (b) speed, (c) virtual length obtained from the speeds, (d) segments of constant length in the virtual path and (e) remapping of the segments on the real path.

Rotation interpolation

Once all the segments in the final path have been defined, the rotation of the segment to define the final path is linearly interpolated between the rotations defined by the user (see Fig 11).

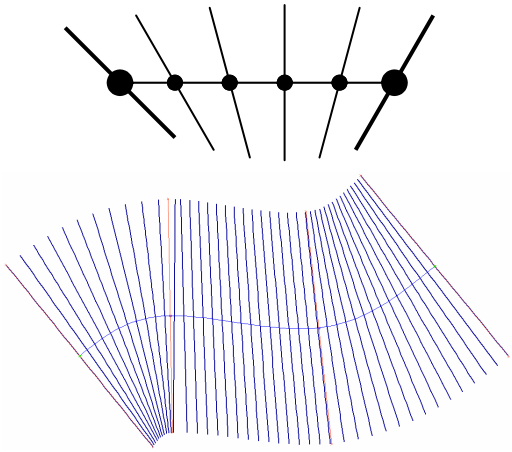


Figure 11. Interpolation of the rotations between the lines defined by the user.

The final interpolation lines do not need to pass exactly through the control points, as shown in Fig. 12. If the final segment subdivisions do not match the control points, the obtained rotation is interpolated between the closest control points.

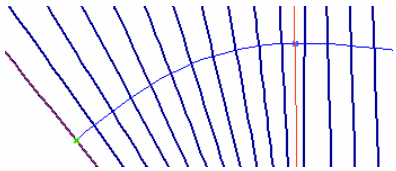


Figure 12. Red line defined by the user, blue lines are interpolated (detail from Fig. 11).

6. SAMPLING

After the transformation is defined, there are two methods for obtaining the final warped image: quadrilateral subdivision or scanline sampling.

Quadrilateral subdivision

The image transformation can be defined as the mapping from the source to the target spaces (see Fig. 4). As the mesh that defines completely the source space is already done, it is straightforward to map every quadrilateral from this mesh to the regular grid in the target space.

There are two main techniques to map a quadrilateral to another quadrilateral: bilinear and perspective [Hec89]. Due to the particular characteristics of this work, as the quadrilaterals are contiguous and can differ in size and shape, the bilinear mapping has been chosen (see Fig. 13).

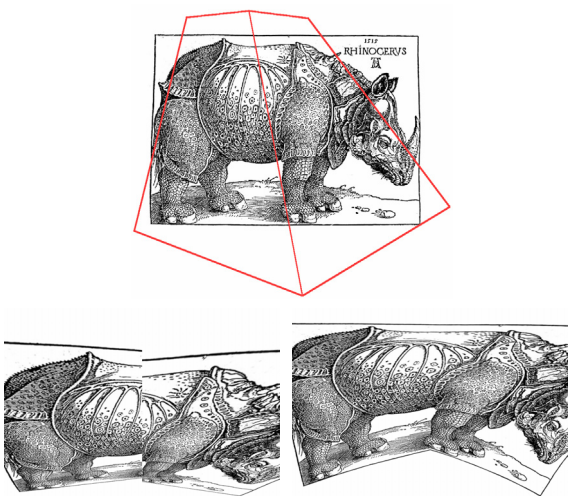


Figure 13. Up: original quadrilateral subdivision. Down-left: perspective transformation. Down-right; bilinear transformation.

The final mesh can have some concave quadrilaterals, but the bilinear transformation can manage it without problems. For example, if the source quadrilateral crosses itself, defining a singular point, the bilinear transformation simply applies that point from the source space to one line in the target space (see Fig. 14).

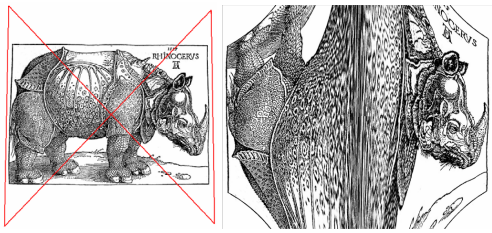


Figure 14. Bilinear concave deformation: original quadrilateral (left) and bilinear transformation result (right).

Scanline sampling

In the original space the transformation is defined by sweeping a straight line, so it is possible to use another mapping method that takes advantage of it. In this method, every line from the source space can be directly mapped to the target space.

A number of lines equal to the number of vertical pixels in the final image must be generated to define the source line distribution to be sampled, with the same algorithm described in Section 5 (Mesh Generation). The obtained lines are directly mapped into every pixel row of the final image. See down part of Fig. 11 for one possible approximation of the source lines to be sampled.

This sampling process approximates how the real analog technique works.

7. RESULTS

This technique can lead to very expressive images, as shown in Figures 15, 16 and 17.

Artists can now add this new digital technique to obtain copy-art simulated images, and it provides a very easy-to-use and intuitive way to warp images.

8. FUTURE WORK

Transformation definition

This technique can easily be extended to the use of straight sweeping lines to any arbitrary sweeping curve. The definition of the deformation would add the possibility of edition of the current lines to transform them into any curve, and the mesh and sampling calculations would interpolate between lines. If the new process is extended to curved lines, the scanline transformation must map a curved line from the source space to a straight one in the target space.

This improvement can lead to a general warping method, similar to the one described in [Lin05].

Mesh modification

The calculation of an intermediate mesh between the parameter specification and the final image warping generation can give the possibility to add more modifications to the transformation. If a set of mesh modification tools is offered to the user, some minor or major changes in the mesh would be applied after the mesh generation, giving the user more possibilities.

Use on animations

In this paper the technique is used on a static image, but if an animation or video is used, it can lead to interesting effects, for example a simulation of a photo-finish camera. See [Gol04], [Davi] or [Davie] for more details.

9. ACKNOWLEDGMENTS

Our thanks to FRACTAL GRAPHICS S.L. company for supporting this research.

10. REFERENCES

[Ara95] Nur Arad and Daniel Reisfeld. Image warping using few anchor points and radial functions. Computer Graphics Forum, 14(1):35-46, March 1995.

[Ara98] Nur Arad. Grid-distortion on nonrectangular grids. Computer Aided Geometric Design, 15(5):475-493, 1998.

[Barr84] A. H. Barr. Global and local deformations of solid primitives. Computer Graphics, 3(18):21-30, July 1984.

[Bei92] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. Computer Graphics (Siggraph'92 Proceedings) volume 26, pages 35-42, July 1992.

[Bern] Marshall Bern. Bern's Copy Art page. http://www2.parc.com/csl/members/bern/copyart.html.

[Davi] Andrew Davidhazy. Basics of Strip Photography http://www.rit.edu/~andpph/text-strip-basics.html

[Davie] Gareth Davies. Linear or Strip Images gallery. http://tickpan.co.uk/strip/

[Gol04] Michael Golembewski. Scanner cameras. http://www.interaction.rca.ac.uk/alumni/02-04/michael/ScannerCamera/index.html 2004

[Gom99] Jonas Gomes, Lucia Darsa, Bruno Costa, Luiz Velho. Warping and Morphing of Graphicsl Objects. Morgan Kaufmann Publishers Inc, 1999.

[Hec89] Paul S. Heckbert. Fundamentals of texture mapping and image warping master's thesis, UCB/CSD 89/516. Technical report, CS Division, EECS Dept, UC Berkeley, May 1989.

[Lin05] Jordi Linares, Jordi Santonja and David Cuesta. Un nuevo método de deformación de imágenes mediante descomposición. I Congreso Español de Informática CEIG2005. Granada 2005.

[MIDE] Museo Internacional de Electrografía. Cuenca. http://www.mide.uclm.es/

[Mil02] Tim Milliron, Robert J. Jensen, Ronen Barzel, and Adam Finkelstein. A framework for geometric warps and deformations. ACM Transactions on Graphics, 21(1):20-51, January 2002.

[Pil02] Maurizio Pilu. Undoing Paper Curl Distortion Using Applicable Surfaces. CVPR (1), pp. 67-72, 2001

[Seu96] Seungyong Lee, George Wolberg, Kyung-Yong Chwa, and Sung Yong Shin. Image Metamorphosis with Scattered Feature Constraints. IEEE Transactions on Visualization and Computer Graphics, 2(4):337-354, December 1996.

[Wol90] George Wolberg. Digital Image Warping. IEEE Computer Society Press. 1990

[Wol95] George Wolberg and Robert C. Loce. Inverting input scanner vibration errors. International Conference on Image Processing. October 1995.

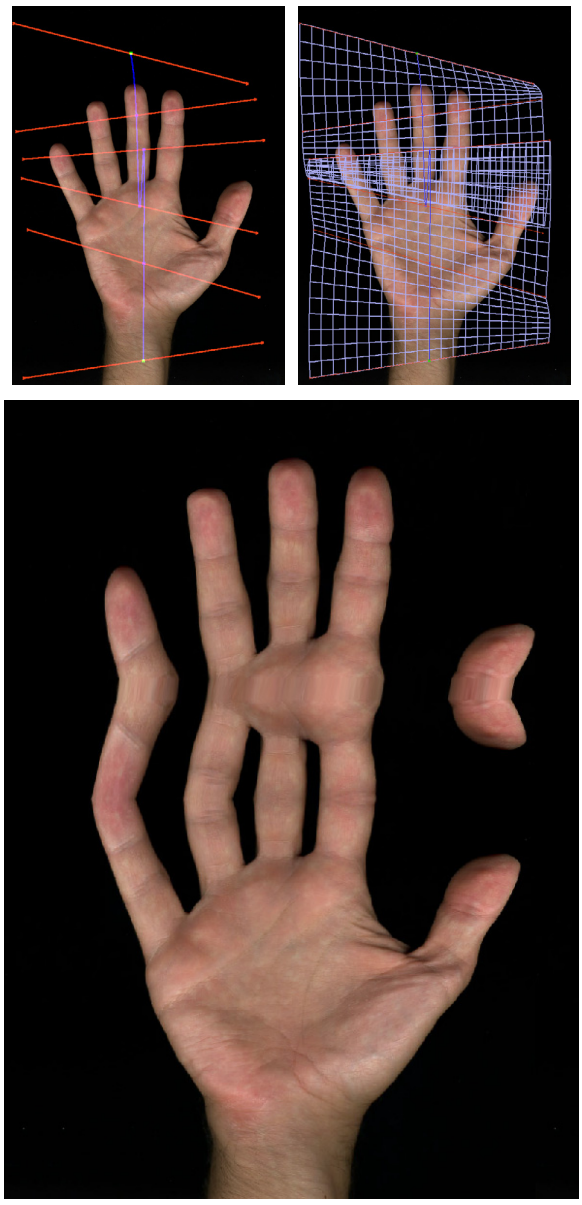
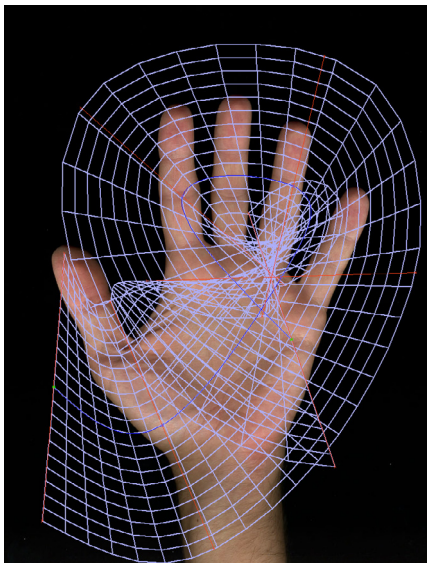


Figure 15. Simple deformation of a hand. Up-left: traced lines, Up-right: mesh, Down: resulting image.



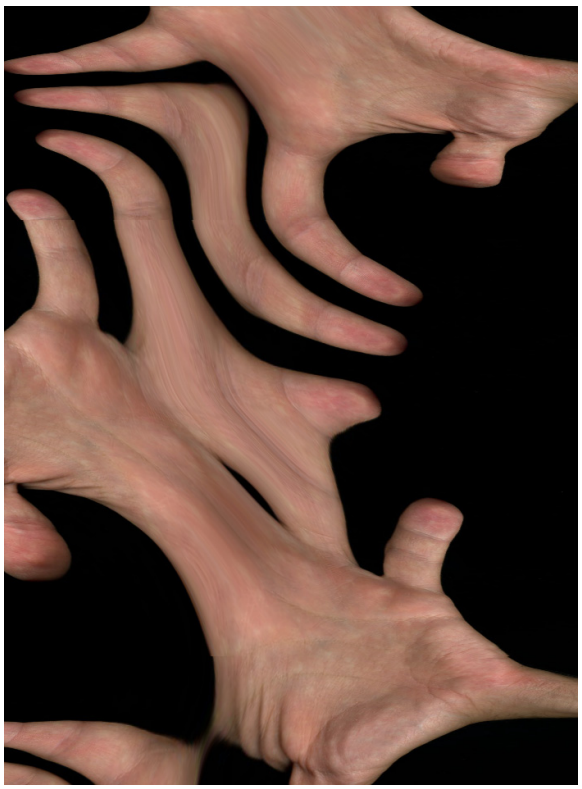
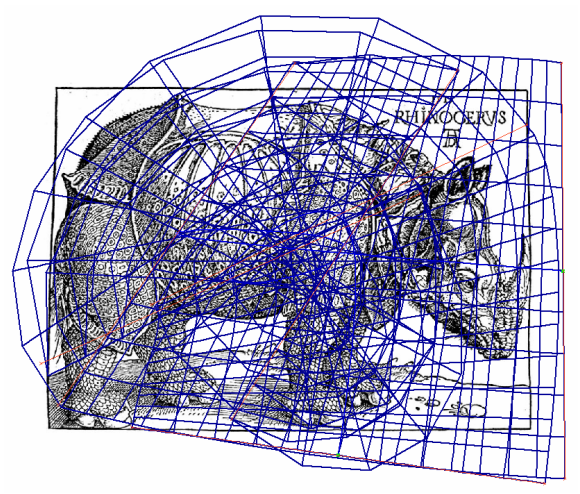


Figure 16. Complex deformation of a hand.



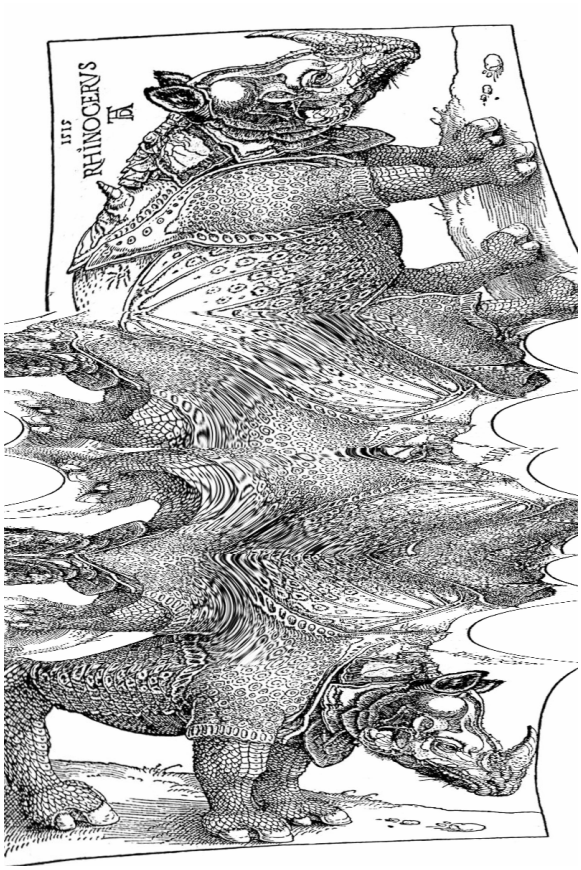


Figure 17. Complex deformation of Albrecht Dürer engraving 'The Rhinoceros', 1515.

Image-based Real-time Hatching of Scene Traveling

Jiajun Bu Wen Yan Chun Chen Mingli Song
College of Computer Science
Zhejiang University
P.R.China 310027, Hangzhou
bjj@zju.edu.cn yanwen@cad.zju.edu.cn {chenc,brooksong}@zju.edu.cn

ABSTRACT

Real-time rendering of a complete 3D scene with hatching strokes is an important direction in NPR field. In this paper, a comprehensive solution is presented to render complicate scenes with pen-and-ink style in real time. With the help of powerful programmable graphics hardware, our real-time system includes many features as hatching, continuous tones, silhouettes, and shadows. We build our approach in image-space, while allowing for the stroke directions and frame coherence.

In our method, various features of pen-and-ink drawings are derived from 3-D information through multi-pass rendering. After synthesis, the desired shading tone is achieved by mapping preprocessed stroke textures to the screen. As a tip for saving texture memory, we prepare a few stroke textures with only certain tones and directions, and compose requisite stroke types in real time. Furthermore, we develop some forms of "indication" to convey the impression of a texture without drawing every single stroke, which makes the result look more natural and art-stylized.

Keywords

Non-photo realistic rendering, real-time hatching, texture indication, G-buffer

1. INTRODUCTION

Pen-and-ink drawing is an important form of pictorial representation [Art77] and an effective way to convey lighting, directions and texture properties. It turns to be a key research branch in non-photorealistic rendering which brings lots of art-like styles. Several features of current GPU-the programmable graphics hardware, can be used to facilitate the art-like real-time rendering, such as large texture capacity, screen texture, multipass rendering, per-pixel process, etc. Currently, many pen-and-ink systems aim to add more features to their renderings while maintaining high frame rate. Most of these systems prefer stroke textures that keep series styles of stroke hatching rather than generating every single stroke in real-time because it is too time-consuming.

In order to generate the pen-and-ink drawing automatically, two types of approaches have been evolved: pen-and-ink drawing in object space and that in image space. The former, called object-based method, attaches stroke textures to polygonal surfaces and render scenes in the traditional rendering pipeline. The latter, called image-based method, parses the scene to get information like tone, silhouette and texture properties and project stroke texture to screen space to achieve different features.

Both these two approaches have merits and demerits. In object-based methods, frame-to-frame coherence can be achieved easily by attaching the stroke texture to scene objects. In this way, strokes can always move with their corresponding objects, and therefore temporal coherence is preserved. However, such methods cannot simulate certain expressive drawing styles conveniently, such as allowing strokes to cross boundaries. Besides, object-based methods require 3D geometry with proper texture coordinates and consequently cannot be applied to images or videos. Moreover, strokes in the screen space are preferred to be independent of the object scale and orientation, namely, uniformly distributed, which is hard to achieve in object-based approaches.

On the other hand, image-based methods overcome many restrictions of object-based methods while introducing some new challenges. They are never fettered by the boundary problem. With

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

image-based technique, strokes can cross polygon edges naturally and be uniformly distributed. However, image-based methods suffer from the undesirable "shower door" effect, which seems that strokes "stick" in screen space independent of object movement. Another problem is that the image-based methods are difficult to control stroke directions to represent scene objects' shapes as ideally as the object-based methods do.

In this paper, we present a novel image-based system that not only preserves the image-based advantages but also adds control to stroke directions as well as reducing "shower door" effect. Our system applies multipass rendering and makes plenty use of G-buffer to achieve desired drawing features. The problem of "stroke direction" mentioned above is tackled by introducing UV map, which stores the pairs of principal directions of each pixel. UV map, produced in rendering pass, is conveyed as a kind of G-buffer for following step to adjust the stroke direction. The direction angle in a certain range is resolved into a same degree in order to avoid too dense variation which usually causes strokes to be severed. Also, by adoption of directions, the "shower door" problem can be weakened because strokes always changes with object's shape and orientation according to view transformation. Moreover, we bring in the idea of indication to enhance the artistic effect of hatching which can also fortify the tone variation and outlines.

The remainder of this paper is organized as follows. In Section 2, we review previous work. The novel rendering algorithm is described in Section 3. A discussion of the implementation and results follows in Sections 4 and 5. The paper ends with a short conclusion and an outlook on future work.

2. Previous Work

Our work aims to render 3D scenes in real time with stroke textures to represent tone, silhouettes, shadows and etc, while avoiding the monotony stroke direction and spatial discontinuity. Related work includes automatic generation of pen-and-ink illustration and silhouettes from 3D scenes, real-time hatching.

Off-line rendering:
Lots of previous work focused on generating high-quality pen-and-ink drawing in an off-line process. Some aimed to create still images of 3D scenes [Deu00, Sai90, Win94, Win96, Sal97, Elb99, Sou99a, Sou99b, Her00]. Saito and Takahashi [Sai90] post-process the rendered framebuffer and overlay image-space strokes. Winkenbach and Salesin [Win94], and Salisbury et al. [Sal97] introduce prioritized stroke textures, which can indicate both texture and tone. Despite the great advance in processing performance, the sheer stroke-drawing hinder these methods from running in real-time.

Real-time Outlines:
Outlines can be generalized as silhouettes, boundaries and creases. They are widely used in various kinds of NPR drawings. Several previous approaches tried to generate outlines in interactive rate [Elb99, Goo99, Her99, Mar97, Nor00, Ras99, Her00, Ras01]. These works can also be divided into two ways: image-based and object-based. The image-based methods often use rendered images as input and are limited by images' resolution. But they are more efficient, easy to implement and sufficient for most projects. The work of Gooch et al. [Goo99] and Raskar [Ras99] can be a typically representation of image-based approach. Object-based approaches aim to produce more precise outlines which are more suitable for additional processing. Hertzmann and Zorin [Her00] create high-quality silhouettes based on geometric duality. Raskar [Ras01] has developed a hardware-support approach to generate outlines by introducing new polygons with only polygonal vertices as input.

Real-time hatching:
Durand et al. [Dur01] create hatched images from photographs in real-time using hardware acceleration to perform anti-aliased threshold. Markosian et al. [Mar97] introduced a simple hatching style indicative of a light source near the camera, by scattering a few strokes on the surface near (or parallel to) silhouettes. Elber [Elb99] shows how to render line art for parametric surfaces in real time; he renders objects by choosing a fixed density of strokes on the surface. Lake et al. [Lak00] describe an interactive hatching system with stroke coherence in image space.

Freudenberg [Fre01a] builds mipmap texture with uniform lines while maintaining the frame-coherence in blending. Praun et al. [Pra01] suggested tonal art map (TAM) for real-time hatching. This algorithm maintains frame-to-frame coherence using a "stroke nesting property", where strokes on a TAM image appear in all the darker images of the same resolution and in the higher resolution images of the same tone. Praun et al. [Pra02] improved their work by providing control of tone to avoid blending and aliasing artifacts in original system. Fung [Fun03] extends the TAM approach and enable generating TAM images of arbitrary textures. Freudenberg's approach [Fre01b, Fre04] express different halftone patterns with stroke textures and implement fast halftone rendering with pixel shading hardware.

3. OUR NOVEL APPROACH

In our pen-and-ink drawing system, the basic features are achieved, like desired tone, outlines and shadows. In the meanwhile, we try to describe the shape of scene objects by fine control of the stroke direction

and give the viewer some indications to avoid monotony. The tone aims at simulating the light shading and can be achieved by composition of TAM (Tonal Art Map) [Pra01] which are generally texture groups of continuous tone and resolution. Because our process is image-based, we need only the finest resolution of the TAM textures. With the aid of G-buffer, 3-D scenes are rendered through several passes to acquire different features. And the indication is generated based on the tip that areas around edges should be paid more ink than the broad unanimous areas which are usually the center of a large surface.

Tone Expression

The terms "value" and "tone" have the same meaning when referred to the amount of visible light reflected from a point to the viewer's eyes. We can use arbitrary lighting model to evaluate the tone at each pixel. The desired tone can be achieved by controlling the ratio of black ink and white paper and we need construct a sequence of hatching images with discrete tones. Abiding by the nesting principle of TAM, our stroke texture series consists of 6 levels of tone in single resolution, as illustrated in Figure 1. This ensures that strokes in the image of lighter tone will also appear in the image of darker tone and therefore temporal coherence (continuity in tone) can be maintained. Actually we only prepare three textures of lighter tone and compose the remainder hatching textures with preceding ones. This strategy can lend economy to texture memory and flexibility to direction control. Since automatic generation of TAM textures is not the goal of this paper, we just use the finest level of TAM textures from Praun's work [Pra01]. In final rendering process, each pixel will be sampled with blend between two textures of consecutive tones.

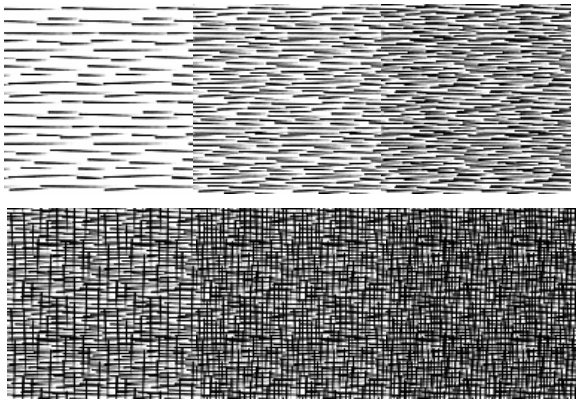


Figure 1. Tonal Map Series. The nether 3 are composed from upper 3 with different rotation angle. Here are only the upright directions.

G-buffer Utility

G-buffer was put forward by Saito and Takahashi [Sai90]. It is a conception that the 2D data structure can be coded to store 3D information. In the rendering process, certain 3D information can be stored into a texture image, each pixel of which should record relevant 3D surface's certain kind of value. The powerful pixel shader can do us a favor to store desired 3D information in RGBA color channels. G-buffer can include many kinds of 3D properties, such as Z buffer, normal buffer, material buffer, shadow buffer, and etc. The following buffers are employed in our system:

- Z buffer: each pixel's corresponding vertex's depth information. Need one float storage.
- Normal buffer: each pixel's corresponding vertex's normal value. Need three-float storage.
- Shadow buffer: shadow map in the view from the light point for each light. The depth value from light point to nearest pixel. Need one float storage.
- UV buffer: principal directions of each pixel recorded as two rotation angle from x-axis or their sine/cosine value. Need at least two-float storage.

To produce these buffers, the rendering process go through several passes and in some cases, two kinds of buffers can be acquired in single pass. Accordingly, two buffers can share one texture image as storage in separate color components. The detail of the rendering process will be mentioned in the Section 4.

Using the G-buffer mentioned above, we can easily generate various features for pen-and-ink drawings. All these features are well developed for many different methods both in image space and object space. For the sake of real-time rendering, we adopt the most effective approaches that can be implemented easily. The following will briefly review these classical methods.

Outline: For polygonal meshes, outlines at silhouettes, ridges, valleys are the key edges for their shape. The silhouette edges consist of edges that are shared by both back-facing polygons and front-facing polygons. A ridge is a crease edge whose dihedral angle between adjacent polygons is less than a threshold while a valley's dihedral the angle is greater than a threshold.

To generate the outline, normal and depth buffers are transferred to a pixel shader which is a filter for postprocessing. For this filter, all we need is to sample pixels around our current pixel and make a comparison between the calculated value and threshold. When detecting silhouettes, it just needs to

check whether the z value of each sampler's normal is of the same sign. When detecting creases, we apply Sobel filter as our convolution matrix to the samplers. Let $I(x, y)$ be a grayscale image. Edge images of I are computed by discrete 2D convolution:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$I_x(x,y) = I(x,y) \otimes S_x, \quad I_y(x,y) = I(x,y) \otimes S_y$$

The optimistic estimation of gradient can be measured as:

$$I_{mag}(x,y) = \sqrt{I_x^2(x,y) + I_y^2(x,y)}$$

Actually, when implemented, even I_x+I_y will be acceptable. And the edge can be detected by the equation below:

$$Edge(x,y) = \begin{cases} 1 & I_{mag}(x,y) \geq T \\ 0 & I_{mag}(x,y) < T \end{cases}$$

Shadow: The approach of shadow map [Seg92] is very suitable for our G-buffer framework. It is a two-pass algorithm. In the first pass, depth map or shadow map can be rendered as mask in view of light's point and naturally stored in G-buffer. Each pixel of the map records the depth of closest pixel to the light. In the second pass, the scene will be rendered from the eye's point while each rasterized fragment's XYZ position is determined relative to the light in order to match the frustum used to create the shadow map. If fragment's light position Z is greater than the depth value at light position XY in the depth map, this fragment is shadowed. Shadow map is stored as a kind of G-buffer.

Indication

The idea of adding some indication is most inspired by the work of Winkenbach who listed indication as one of pen-and-ink drawing's principles. In merits, "Indication" lends economy to an illustration, and also makes an illustration more impressive by engaging the imagination of the viewer rather than revealing everything. Winkenbach aimed to produce static drawings, so he can employ lots of user interaction and make excellent effect. But in real-time implementation we can only introduce very little indication by automatically composing indication map. Our implementation involves two measures.

The indication fields should be depicted with fewer details, and therefore the basic principle is to generate indication in the area far from outlines. Our first measure borrows the idea "field" [Bei92] to evaluate indication. In Winkenbach's work, user distributes some "detail segments" along silhouettes on prepared image and calculates a field $w_1(x, y)$ for each pixel according to these segments. In this calculation, l indicates the nearest "detail segment" labeled by user and a_1, b_1, c_1 are used to adjust the weight.

$$w_1(x,y) = (a_1 + b_1 \times dis((x,y),l))^{-c_1}$$

Without interactively placing "detail segments", we have to generate a rough field map for every frame. An easy approach to achieve the fast distributing requirement is to produce a "General Field Map (GFM)", and attach this GFM to each plane as texture. GFM should be preprocessed according to $w_1(x, y)$, simply with its edges indicating l. The field maps will transform with the scene objects and the indication map of each frame would be composed by them. Figure 3 can illustrate the main idea.

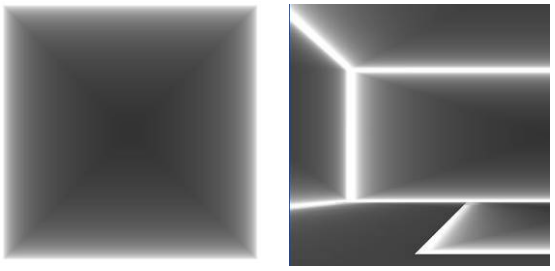


Figure 3: (a) General Field Map: produced according to $w_1(x, y)$, and l indicates the edge of the square map. (b) Composed Field map: Indication appears in dark areas while the lighter the more details to appear.

Without loss of generality, the square GFM can apply to various polygons by adjusting their texture coordinates. This method can work pretty well when the scene is built up by planes connected edge by edge. However, when it comes to curving surface and the intersection of two surfaces, this method seems a little helpless. The area of intersection can not be totally weighted with detail by indication of GFM. In implementation, we can avoid intersection of planes by incising the big plane into small pieces at the place of intersection. But it is not a generic approach and lead to a heavy workload on modeling.

Our second measure deals with the indication by silhouette. In normal depth map, at each pixel, we uniformly sample nearby pixels around it with a certain distance R. Then the difference between central and nearby sampling pixels of normal and depth map can be calculated and compared to a threshold to detect whether the two pixels are separated by silhouettes. And the field of the central pixel can be weighted by:

$$w_2(x,y) = (a_2 + b_2 \times field(N(x,y,R)))^{-c_2}$$

$$field(n) = \alpha / (n^{\beta} + \lambda)$$

$N(x, y, R)$ equals the number of the nearby pixels apart from central pixel by silhouette, where R is the radius of sampled pixel from center. And distance of the current central pixel from silhouette can be weighted by a *field(n)*. This approach can approximate field weight limited to the area of distance R from silhouettes. The precision of field can be enhanced by increasing the sampling pixel number whereas could be very time consuming. Then we have to weigh a tradeoff between precision and running efficiency.

By integrating the above two measures, we can cover most of the high-weight fields (detailed) and approximate the indication area successfully. The overall field weight can be evaluated by:

$$w(x,y) = a \times w_1(x,y) + b \times w_2(x,y)$$

Direction Control

In the ideal state, strokes are supposed to follow the principal direction in each pixel. However, when scaled or rotated, a polygonal plane will be distorted and UV direction within different areas of the plane can never be kept in parallel. This is not a great problem in object-space because the stroke textures could be attached to facets along UV direction and these textures are being transformed with objects. Nevertheless, strokes can be seriously compressed and aliased when facets are nearly perpendicular to the screen. Even TAM textures in object-based methods can not perfectly settle this problem.

In image-space, we only need to concentrate on the UV direction problem as the strokes can be uniformly distributed through the whole screen space and they will not be distorted in size. In our algorithm, UV direction has been designated in the object space. When rasterized, in each pixel, UV direction will be transformed to the angle deviated from the X axis and be stored in G-buffer. Then in texture mapping, each pixel's coordinates should be applied to rotation matrix according to the UV map. In the UV coordinates' transformation, serious aliasing occurs as the stroke direction changes continuously. To get local coherence, we set a threshold T so that the direction can only change in leap. This can be simply implemented as:

$$angle = sign(angle) \times \lfloor abs(angle) / T \rfloor \times T$$

The process is demonstrated in the Figure 4. To show the direction transformation clearly, only one tone has been used in the following illustration. And the result image with direction and tone control is shown in Figure 5

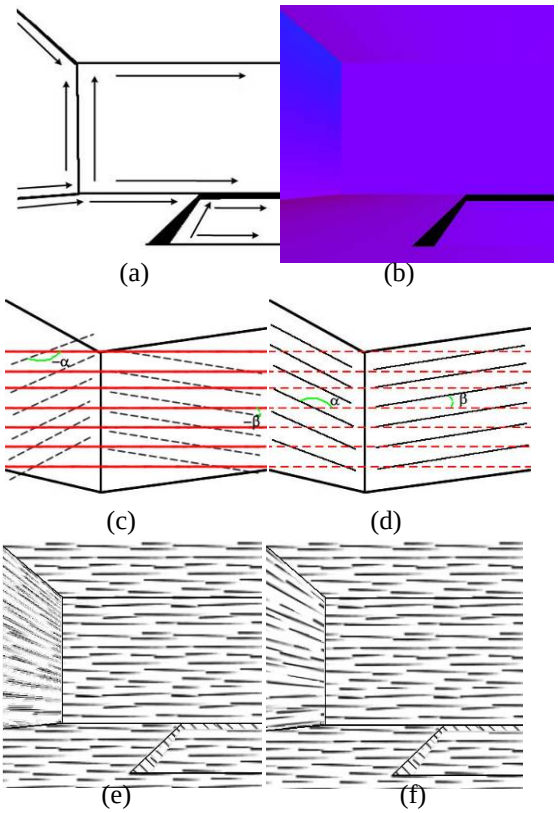


Figure 4: Stroke direction control. (a) UV direction vectors attached to each plane transform with coordinates transformation. (b) Record UV rotation angle to G-buffer. (c)&(d) adjust the texture coordinates to sample stroke texture. Red lines indicate stroke textures direction. Black lines indicate UV directions. To be noticed, texture coordinates should be rotated an inverse angle of UV direction. (e) Stroke directions without the threshold subsection. (f) Stroke directions with threshold subsection.

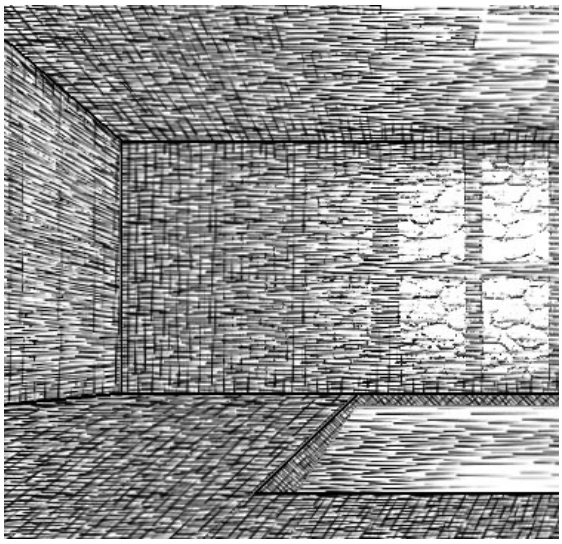


Figure 5: Hatching with stroke direction: This figure also includes tones and outlines.

4. IMPLEMENTATION

Our image-based hatching system takes advantage of GPU hardware ability, and is able to render scenes in a few passes to achieve multiple features in real-time. Optionally, we could skip some of features to pursuit high frame rate or to make one of them more distinct. The choice lies on the user. Figure 6 illustrates the relation between G-buffer and features. The composed images are illustrated in Figure 8.

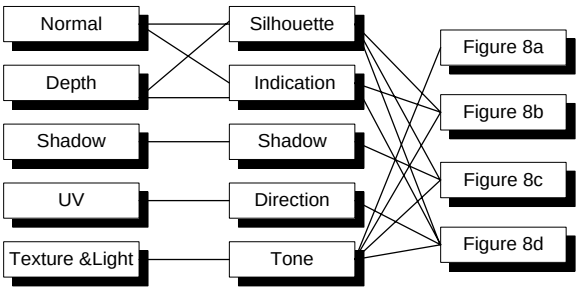


Figure 6: Relation between G-buffer and features

The implementation is based on C++ and OpenGL. The framework runs with a 1.2 GHz CPU and Radeon 9550 graphics card which can provide us Pixel Shader2.0 functionality. Many tools can be used to record G-buffer attributes as texture images, like "RenderTexture", "PBuffer" and etc. We choose hardware supported "Frame Buffer Object" which is proved to be the fastest. To add all the mentioned features, we need our scenes go through 6 passes. In each pass, special shader is designed for desired rendering purpose. The following framework pipeline (in Figure 7) can illustrate the whole process.

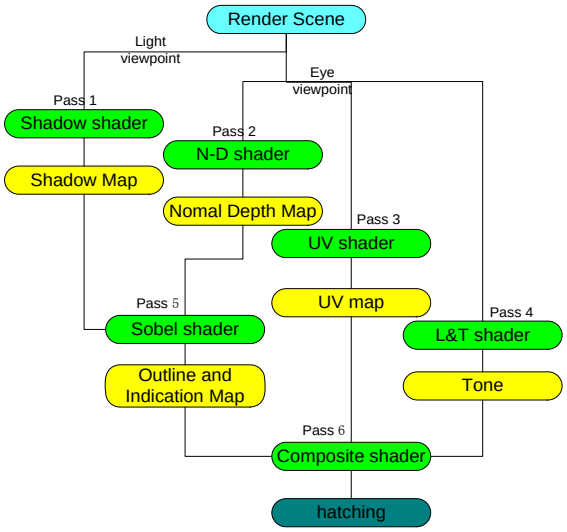


Figure 7: Multipass Rendering Pipeline. Blue component indicates input data. Green ones indicate rendering shaders and yellow ones indicate G-buffer after each pass.

With the six passes, a comprehensive balance strategy is considered in our approach. We try to compress two or more shaders into one pass while avoiding the overload in single shader or possible bottle neck. The shadow buffer can be integrated into the Outline&Indication buffer, each of which occupies one color channel originally, so as to reduce sampling computation in Pass 6. In Pass 2, RGBA channels can not be shared with other buffer because they are fully occupied by Normal&Depth buffer. In Pass 3, direction buffer is stored as (cosU, sinU, cosV, sinV) in RGBA. As for Pass 4, the edge-detection on textures is performed since the textures themselves also have complicate paintings and we would like their edges to be accentuated. And tone value in Pass 4 needs only one channel if the hatching image is in grey level without color texture.

5. CONCLUSION and FUTURE WORK

In this paper, we present a system for image-based hatching that can efficiently render 3-D scenes in real-time by GPU. The power of GPU brings facility to the implementation of our rendering algorithm and makes it possible that lots of features can be integrated into rendering in real-time. The main contribution of this paper is that of simulating the idea of indication in pen-and-ink paintings and stroke direction controlling in image space. Both of them can make the painting more natural, reveal objects' shape and weaken the "shower door" effect to some extent,.

However, our approach of stroke direction control may cause the stroke to be ruptured and lose spatial coherence when directions vary sharply in small area. Therefore, our future work aims to maintain the full coherence while revealing the stroke direction. In addition, our approach to generate indication is not so effective in complicated scenes. The implementation of the "field map" could be improved by using the ability to write texture in pixel shader in up-to-date GPU. Then many of samplings are not necessary when approximating the indication field, which costs lots of computation. Furthermore although the "shower door" effect is reduced in our approach, further effort is needed to eliminate it thoroughly.

6. REFERENCES

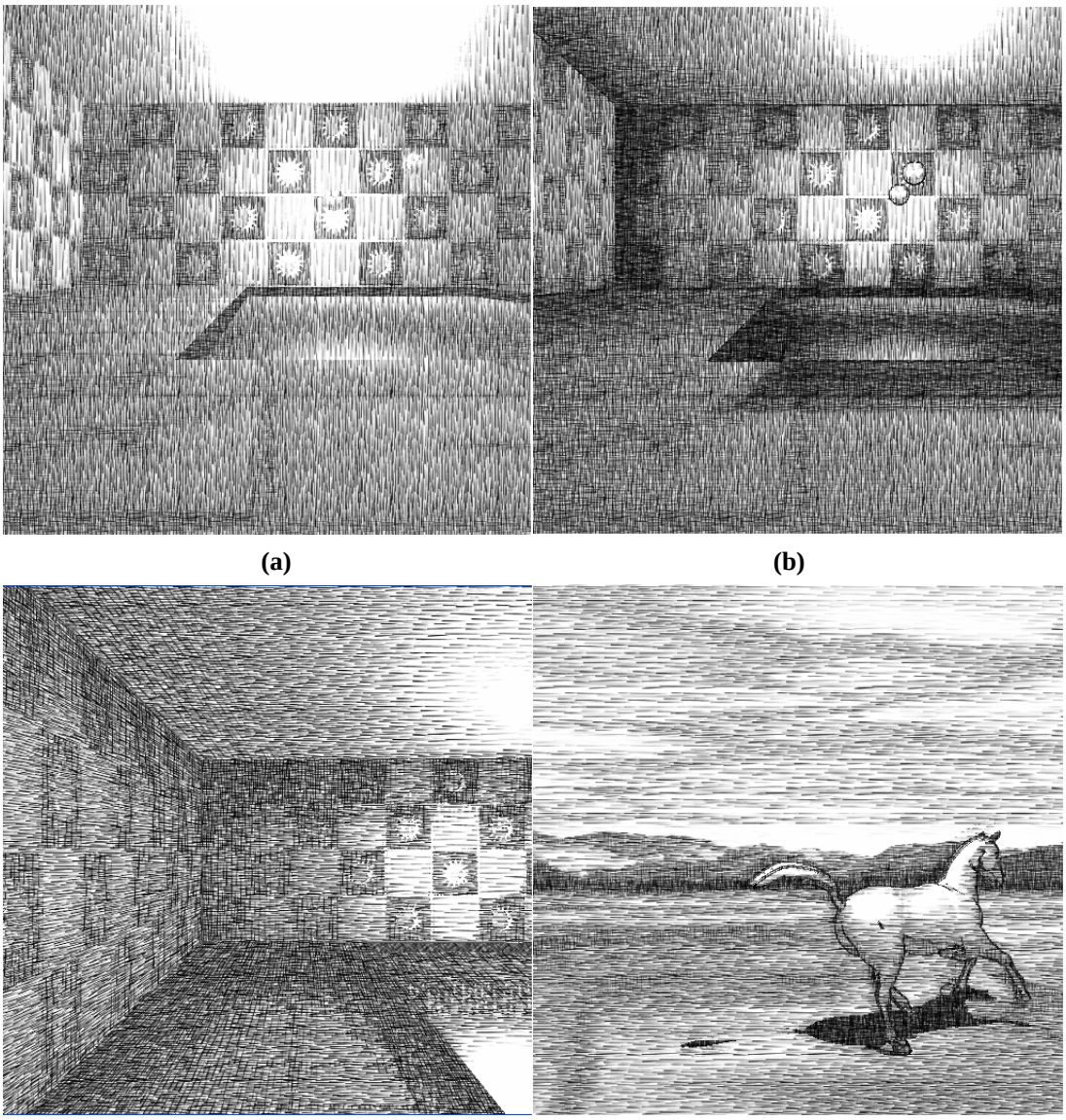
[Art77] Arthur, L. Guptill. Rendering in Pen and Ink. Watson-Guptill Publications, New York, 1977.

[Bei92] Beier, T. and Neely, S. Feature-based image metamorphosis. Proceedings of SIGGRAPH '92 In Computer Graphics 26, 2 (July 1992), 35–42.

[Sai90] Saito, T. and Takahashi, T. Comprehensible Rendering of 3D Shapes. Proceedings of SIGGRAPH 90, pp. 197–206.

[Win94] Winkenbach, G. and Salesin, D.H.

Computer-Generated Pen-and-Ink Illustration. Proceedings of SIGGRAPH 94, Computer Graphics, Annual Conference Series, pp. 91–100.
[Sal97] Salisbury, M.P., Wong, M.T., Hughes, J.F., and Salesin, D.H. Orientable Textures for Image-Based Pen-and-Ink Illustration. Proceedings of SIGGRAPH 97, pp. 401–406.
[Deu00] Deussen, O., and Strothotte, T. Computer-Generated Pen-and-Ink Illustration of Trees. Proceedings of SIGGRAPH 2000, 13–18.
[Elb99] Elber, G. Interactive Line Art Rendering of Freeform Surfaces. Computer Graphics Forum 18, 3 (September 1999), pp. 1–12.
[Sou99a] Sousa, M.C., and Buchanan, J.W. Observational Model of Blenders and Erasers in Computer-Generated Pencil Rendering. Proceedings of Graphics Interface'99, 157 – 166.
[Sou99b] Sousa, M.C., and Buchanan, J. W. Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models. Computer Graphics Forum 18, 3 (September1999), pp. 195–208.
[Win96] Winkenbach, G., and Salesin, D.H. Rendering Parametric Surfaces in Pen and Ink. Proceedings of SIGGRAPH 96, pp. 469–476.
[Her00] Hertzmann, A., and Zorin, D. Illustrating Smooth Surfaces. Proceedings of SIGGRAPH 2000, Computer Graphics, Annual Conference Series, pp. 517–526.
[Goo99] Gooch, B., Sloan, P.-P. J., Gooch, A., Shirley, P., and Riesenfeld, R. Interactive Technical Illustration. 1999 ACM Symposium on Interactive 3D Graphics, pp. 31–38.
[Mar97] Markosian, L., Kowalski, M.A., Trychin, S.J., Bourdev, L.D., Goldstein, D., and Hughes, J.F. Real-Time Nonphotorealistic Rendering. Proceedings of SIGGRAPH 97, pp. 415–420.
[Nor00] Northrup, J.D., and Markosian, L. Artistic Silhouettes: A Hybrid Approach. Proceedings of NPAR 2000, pp. 31–38.
[Ras99] Raskar, R., and Cohen, M. Image Precision Silhouette Edges. 1999 ACM Symposium on Interactive 3D Graphics, pp. 135–140.
[Ras01] Raskar.R Hardware Support for Non-photorealistic Rendering. 2001 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, pp. 41-47
[Her99] Hertzmann, A. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. SIGGRAPH Course Notes. 1999.
[Lak00] Lake, A., Marshall, C., Harris, M., and Blackstein, M. Stylized Rendering Techniques for Scalable Real-Time 3d Animation. Proceedings of NPAR2000, pp.13–20.
[Pra01] Praun, E., Hoppe, H., Webb, M., and Finkelstein, A. Real-Time Hatching. Proceedings of SIGGRAPH 2001, Computer Graphics, Annual Conference Series, pp. 579-584.
[Pra02] Webb, M., Praun, E., Finkelstein. A., and Hoppe, H. Fine Tone Control in Hardware Hatching. Proceedings of NPAR 2002. pp. 53-58
[Fre01a] Freudenberg, B., Masuch, M. and Strothotte, T. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. Proceedings of Eurographics 2001, pp. 184-191.
[Fun03] Fung, J., Veryovka, O. Pen-and-ink textures for real-time rendering. Proceedings of Graphics Interface 2003. pp. 131-138
[Fre04] Freudenberg, B., Masuch, M. and Strothotte, T. Real-Time Halftoning: Fast and Simple Stylized Shading. Game Programming Gems 4, Charles River Media, 2004.
[Fre01b] Freudenberg, B. Real-Time Stroke Textures. SIGGRAPH 2001 Conference Abstracts and Applications, pp. 252.
[Seg92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli: Fast shadows and lighting effects using texture mapping. SIGGRAPH 1992, pp.249-252.
[Dur01] Durand, F., Ostromoukhov, V., Miller, M., Duranleau, F. and Dorsey, J. Decoupling Strokes and High Level Attributes for Interactive Traditional Drawing. Proceedings of Eurographics Rendering Workshop 2001, pp. 71-82.



(a) **(b)**

(c) **(d)**

Figure 8: Different combination of features

The Hierarchical Ray Engine [1]

László Szécsi

Dept. of Control Engineering and Information Technology
Budapest University of Technology and Economics
Magyar Tudósok krt. 2., H-1117, Hungary
szecsi@iit.bme.hu

ABSTRACT

Due to the success of texture based approaches, ray casting has lately been confined to performing preprocessing in realtime applications. Though GPU based ray casting implementations outperform the CPU now, they either do not scale well for higher primitive counts, or require the costly construction of spatial hierarchies. We present an improved algorithm based on the Ray Engine approach, which builds a hierarchy of rays instead of objects, completely on the graphics card. Exploiting the coherence between rays when displaying refractive objects or computing caustics, realtime frame rates are achieved without preprocessing. Thus, the method fills a gap in the realtime rendering repertoire.

Keywords: Ray tracing, GPU programming.

1 INTRODUCTION

With high performance hardware designed to support scan conversion image synthesis, most research aims to eliminate time consuming ray casting from illumination algorithms, or to move it to a preprocessing step computing texture maps. However, there are some light transport effects that exhibit inherently recursive behavior, most prominently visible refractive objects, or caustics via multiple reflections or refractions. Accurate maps or transport factor matrices cannot be constructed with a feasible storage requirement. On the other hand, these problems are effectively handled by recursive raytracing or photon tracing, both based on ray casting. Moreover, if we consider eye rays or light rays from small light sources, hitting reasonably smooth objects, the rays to be traced will be coherent, even after multiple reflections of refractions.

In order to make ray casting feasible in realtime applications, it is imperative to make use of the immense computing power of the GPU. One delivering research direction has spawned from the approach of Purcell et al.[6], the impact of which we will briefly evaluate in Section 6. If we are looking for a solution which does not rely on a pre-built acceleration structure, the most important milestone we find is the Ray Engine[2]. Based on the recognition that ray casting is a crossbar on rays and primitives, while scan conversion is a crossbar on pixels and primitives, they have devised a method for computing all possible ray-primitive intersections on the GPU. On contemporary hardware they could achieve processing power similar to the CPU's.

2 PREVIOUS WORK

2.1 The ray engine

As the ray engine serves as the basis of our improved approach, let us reiterate its working mechanism in current GPU terminology. Every pixel of the render target is associated with a ray. The origin and direction of rays to be traced are stored in textures that have the same dimensions as the render target. In every pass, a single ray casting primitive is taken, and it is rendered as a full-screen quad, with the primitive data attached to the quad vertices. Thus, pixel shaders for every pixel will receive the primitive data, and can

also access the ray data via texture reads. The ray-primitive intersection calculation can be performed in the shader. Then, using the distance of the intersection as a depth value, a depth test is performed to verify that no closer intersection has been found yet. If the result passes the test, it is written to the render target and the depth buffer is updated. This way every pixel will hold the information about the nearest intersection between the scene primitives and the ray associated with the pixel. The pitfall of the ray engine is that it implements the naive ray casting algorithm of testing every ray against every primitive. On the other hand, ray casting research has been directed on building effective acceleration hierarchies to minimize the number of actual intersection tests performed [1]. Unfortunately, these results cannot be easily ported to the graphics hardware.

2.2 Exploiting coherence on the CPU

It is relevant to mention that besides hierarchies, methods making use of image-space coherence were also exhaustively researched[8]. Most importantly, we will make use of the motif of decomposing the image space into tiles containing assumedly coherent rays in our algorithm.

2.3 Acceleration hierarchies for the GPU

Foley and Sugerman[3] implemented the architecture foreseen by Purcell et al.[6]. The kd-tree acceleration hierarchy, formerly confined to the CPU, is traversed on the GPU using algorithms not optimal in the worst-case algorithmic sense, but eliminating the need of a stack. This offers a competitive alternative to CPU ray tracing, but it is not directly targeted on real-time applications, and it is ill-suited for highly dynamic scenes because of the construction cost of the kd-tree.

2.4 Approximate ray tracing

Szirmay-Kalos et al. [7] use an improved version of environment mapping to take the origin of reflected or refracted rays into account when reading a cube map of the environment. They have also extended the method to handle the rear refraction on transparent objects. To be accurate, they require the environment and refractive object geometry to be star-shaped, so that all the geometry can be represented in a single cube map. As rendering these cube maps is costly, it has to be amortized to be real-time, so the method is applicable for scenes with low dynamism, or with a single moving object in a static environment.

2.5 Memory-Coherent Ray Tracing

Pharr et al.[5] have developed algorithms that use caching and lazy creation of texture and geometry to manage scene complexity and assure coherent access. The approach is also very effective at improving the performance of the ray engine[2]. However, the focus and possible areas of application are very different from our approach. They discuss how complex scenes can be effectively managed over various levels of conventional storage architectures, where rendering times are in the magnitude of several hours. On the other hand, our method is tailored for current graphics hardware, scenes not more complex than in typical interactive environments like computer games, but achieving real-time rendering frame rates.

3 ACCELERATION HIERARCHY FOR THE RAY ENGINE

CPU-based acceleration schemes are spatial object hierarchies. Although considerable research has dealt with exploiting the coherence between neighboring rays, including longest common traversal sequences [4] and image space interpolation [9], these have not altered the basic approach. That is, for a ray, we try to exclude as many objects as possible from intersection testing. This cannot be done in the ray engine architecture, as it follows a per primitive processing scheme instead of the per ray philosophy. Therefore, we also have to apply an acceleration hierarchy the other way round, not on the objects, but on the rays.

In typical applications, realtime ray casting augments scan conversion image synthesis where recursive ray tracing from the eye point or from a light sample point is necessary. In both scenarios, the primary ray impact points are determined by rendering the scene from either the eye or the light. As nearby rays hit similar surfaces, it can be assumed that reflected or refracted rays may also travel in similar directions, albeit with more and more deviation on multiple iterations. If we are able to compute enclosing objects for groups of nearby rays, it may be possible to exclude all rays within a group based on a single test against the primitive being processed. This approach fits well with the ray engine. Whenever the data of a

primitive is processed, we should find a way not to render it on the entire screen as a quad, but invoke the pixel shaders only where an intersection is possible. An obvious solution is to split the render target into tiles, render a set of tile quads instead of a full screen one, but make a decision for every tile beforehand whether it should be rendered at all. At a first glimpse, this may appear counterproductive, as, apparently, far more quads will be rendered. However, there is a set of issues that disprove concerns.

- The ray engine is pixel shader intensive, and makes practically no use of the vertex processing unit. The number of pixel shader runs, which remains crucial, is by no means increased.
- The high level test of whether a tile may include valid intersections can be performed in the vertex shader. If the intersection test fails, the vertices of the quad are transformed out of view, and discarded by clipping. Moving the vertices out of view does not require any computation, they are simply assigned an outlying extreme position.
- Instead of small quads, one can use point primitives, described by a single vertex. This eliminates the fourfold overhead of processing the same vertex data for all quad vertices, and needlessly interpolating values.

In order to implement this idea, we have to solve two problems. First, for rays grouped in the same tile, an enclosing object should be computed, for which an intersection test is fast. This computation should be performed on the GPU. Second, the data describing these enclosing objects should be accessible to the vertex shader.

The latter problem can be resolved by texture reads in the vertex shader. If data is rendered to a texture, where every texel corresponds to an enclosing object, it can be accessed when processing the appropriate tile. If we reorganize the rendering process, even this vertex texture fetch can be eliminated. Remember that we are rendering a tile for every ray casting primitive, for every possible tile position. Now if we take a tile position, and render all the primitives there at once, the enclosing object information is static. It can be passed to the vertex shader in uniform parameters. In order to do this, we have to read back the texture holding the enclosing object data from the graphics card. However, as it contains only as many texels as many tiles are used (16×16 is typical), this is not an expensive operation.

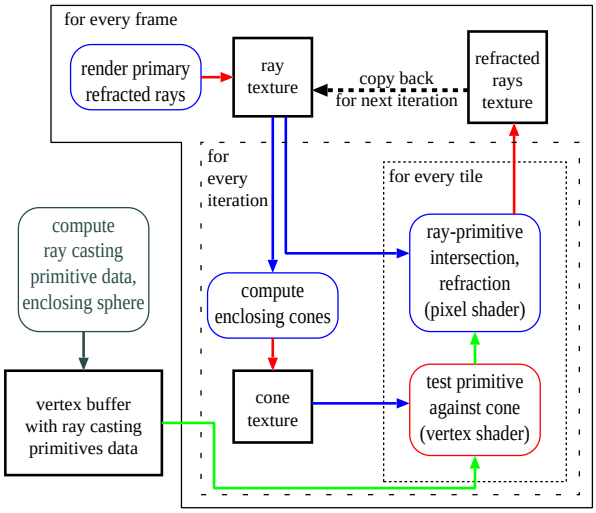


Figure 1: Block diagram of the hierarchical ray engine. Only the initial construction of the vertex buffer is performed on the CPU.

Figure 1 depicts the data flow in an application for tracing refracted rays, using the proposed method. Ray casting primitives are encoded as single vertices, and can be channeled to the shaders as a vertex buffer of point primitives. As a result of the intersection tests, the ray defining the next segment of the refraction path is written to the render target. The process is repeated for pixels, in which the path has not yet terminated. In the beginning of every iteration, an enclosing cone for rays is built for every tile, and stored in a texture. This texture is used in consequent vertex shader runs to carry out preliminary intersection tests.

Note that the cones have to be reconstructed for every new generation of rays, before the pass computing the nearest intersections. They cannot be precomputed on the CPU, and their construction must also be fast and possible to implement on the GPU.

4 CONSTRUCTION OF AN ENCLOSING CONE

We need to be able to perform the intersection test between the enclosing object and the ray casting primitive as fast as possible. At the same time, representation of both the primitives and the ray-enclosing objects must be compact, because primitive data has to be passed in a very limited number of vertex registers, and enclosing

objects must be described by a few texels. One rapid test is the intersection test between an infinite cone and a sphere. Enclosing spheres for all ray casting primitives can easily be computed, and described by a 3D position and a radius. Enclosing infinite cones of rays are described by an origin, a direction and an opening angle.

The infinite enclosing cones must be constructed in a pixel shader, in a pass before rendering the intersection records themselves. Note that in a practical application, the rays to be traced will be different for every frame, and for every level of refraction, so the reconstruction of the cones is also time critical. Therefore, a fast incremental approach is preferred over a tedious one, which could possibly produce more compact results, via, for instance, linear programming. The algorithm goes as follows:

1. Start with the zero angle enclosing cone of the first ray.
2. For each ray
   (a) Check if the direction of the ray lies within the solid angle covered by the cone, as seen from its apex. If it does not, extend the cone to include both the original solid angle and the new direction.
   (b) Check if the origin of the ray is within the volume enclosed by the cone. If it is not, translate the cone so that it includes both the original cone and the origin of the ray. The new cone should touch both the origin of the ray and the original cone, along one of its generator lines.

Both steps of modifying the cone require some mathematics. Let $\vec{x}$ be the axis direction of the cone, $\vec{a}$ its apex, φ the half of the opening angle, $\vec{r}$ the direction of the ray, and $\vec{p}$ its origin.

First, if the solid angle defined by the cone does not include the direction of the ray, the cone has to be extended (See Figure 2). This is the case if $\vec{x} \cdot \vec{r} < \cos\varphi$. Then, the generator direction $\vec{e}$, opposite to the ray direction, has to be found. If $\vec{r}$ is projected onto $\vec{x}$, the direction from $\vec{r}$ to the projected point defines $\vec{q}$:

$$\vec{q} = \frac{(\vec{x} \cdot \vec{r}) \cdot \vec{x} - \vec{r}}{|(\vec{x} \cdot \vec{r}) \cdot \vec{x} - \vec{r}|}.$$

Then $\vec{e}$ is found as a combination of $\vec{x}$ and $\vec{q}$:

$$\vec{e} = \vec{x} \cdot \cos\varphi + \vec{q} \cdot \sin\varphi.$$

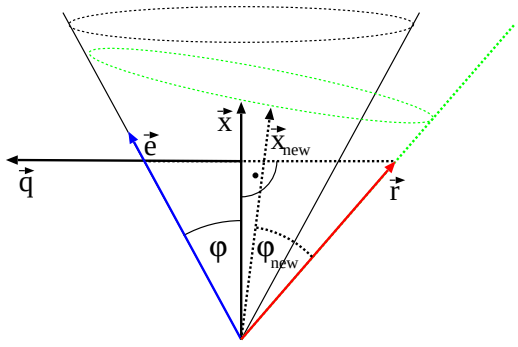


Figure 2: Extending the cone.

The new axis direction should be the average of $\vec{e}$ and $\vec{r}$, and the opening angle should also be adjusted:

$$\vec{x}_{\text{new}} = \frac{\vec{e} + \vec{r}}{|\vec{e} + \vec{r}|}, \qquad \cos\varphi_{\text{new}} = \vec{x}_{\text{new}} \cdot \vec{r}.$$

Given the information we had, which does not include any knowledge of rays already within the cone, we can state that this method computes the cone of minimum opening angle necessary to hold the given infinite semi-line and the cone.

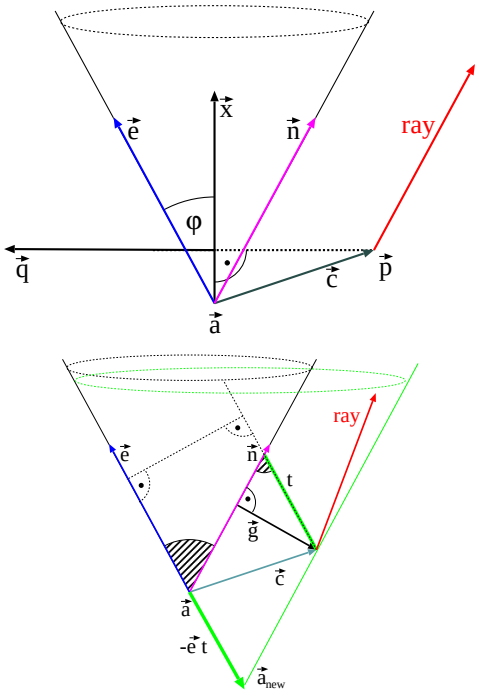


Figure 3: Finding the near and far generators, and translating a cone.

Translating the possibly extended cone to include the origin is somewhat more complicated, but follows the same trail (See Figure 3). First the *near-*

est generator direction $\vec{n}$ and the *farthest* generator direction $\vec{e}$ are found just like before. The vector $\vec{c} = \vec{p} - \vec{a}$ plays the role what $\vec{r}$ had in the previous computation;

$$\vec{q} = \frac{(\vec{x} \cdot \vec{c}) \cdot \vec{x} - \vec{c}}{|(\vec{x} \cdot \vec{c}) \cdot \vec{x} - \vec{c}|}.$$

Like before,

$$\vec{e} = \vec{x} \cdot \cos\varphi + \vec{q} \cdot \sin\varphi, \qquad \vec{n} = \vec{x} \cdot \cos\varphi - \vec{q} \cdot \sin\varphi.$$

We want to translate the cone along generator $\vec{e}$ so that the generator $\vec{n}$ moves to cover the ray origin $\vec{p}$ (Figure 3, on the right). The distance vector $\vec{g}$ between the origin and the nearest generator is found as:

$$\vec{g} = \vec{c} - (\vec{n} \cdot \vec{c}) \cdot \vec{n}.$$

The translation distance t and the new apex position are:

$$t = \frac{\vec{g}^2}{\vec{e} \cdot \vec{g}}, \qquad \vec{a}_{\text{new}} = \vec{a} - \vec{e} \cdot t.$$

Using the two steps in succession, we find a new cone that includes both the previous cone and the new ray, has a minimum opening as a priority, and was translated by a minimum amount as a secondary objective. Of course, knowing nothing about the rays already included in the cone, we cannot state that the computation achieves an optimal result in any way. However, it is conservative and mostly needs vector operations, fitting well in a pixel shader. Furthermore, cone construction is only performed once for every iteration of ray casting.

The computation of enclosing spheres of ray casting primitives is done on the CPU, at the same time when all the scene data is processed. The result is a vertex buffer, in which all ray casting primitives are encoded as vertices. Note that the position value slot can be used for passing the enclosing sphere data, as it will simply be exchanged with the tile position in the vertex shader. The cone intersects the sphere if

$$\varphi > \arccos[(\vec{v} - \vec{a}) \cdot \vec{x}] - \arcsin[r/|\vec{v} - \vec{a}|],$$

where $\vec{a}$, $\vec{x}$ and φ describe the cone as before, $\vec{v}$ is the center of the sphere and r is its radius.

The vertex data describing a triangle primitive may be composed as:

position enclosing sphere center and radius

normal the normal and offset of the triangles plane

texture 0-2 pre-processed triangle vertex position data for fast intersection computation

further texture registers normals at the triangles vertices

further texture registers texture coordinates at the triangles vertices

5 Excluding terminated ray paths

In a typical recursive raytracing problem, we may divide the geometry into ideally refractive or reflective surfaces, and locally shaded ones. We will refer to these groups as ideal and non-ideal surfaces. We start off with a texture of eye and light rays. Our method should follow these rays through multiple ideal reflections and refractions, until the exiting rays do not hit an ideal surface any more. These final rays we get as a result can either be traced against the non-ideal geometry of the scene, or used in any other method like environment mapping or caustics generation.

Firstly, there may be pixels in which no refractive or reflective surface is visible. Furthermore, in every iteration replacing rays with their reflected or refracted successors, there will be rays not arriving on any reflective of refractive surface, producing no output. It is desirable that for those pixels where there is no ray to trace, the pixel shader is not invoked at all. This is achieved using the stencil buffer and early stencil testing. When rendering intersections, every bit of the stencil serves as a flag for a specific iteration. The stencil read and write masks select the flag bit of the previous and current iterations, respectively. Should ray casting fail to hit any object, the stencil bit will not be set, and in the next iteration the pixel will be skipped. With an eight bits deep stencil buffer, this allows for eightfold reflection or refraction to be traced. Further iterations are possible without excluding further terminated paths.

6 RELATION TO OTHER APPROACHES

The ray engine was designed to be a general purpose raytracer, with no preconceptions on what rays there are to be traced. While this generality could be considered a great advantage, current advancements in technology and research make

the ray engine approach obsolete in all but few areas of application. Firstly, real time global illumination is better supported by texture-based methods making use of precomputed maps or rendered environments. Secondly, in offline computations, where ray casting is still essential, ray casting acceleration schemes well known on the CPU will very soon be adopted for the graphics hardware [6]. The acceleration structure must still be built on the CPU, rendering the approach incapable of realtime rendering of dynamic scenes. That practically leaves those two areas for realtime ray casting which our hierarchical ray engine specializes in: visible refractive objects and caustics. Our algorithm can outperform the ray engine by making use of the coherence of rays in these particular problems.

Figure 4: Cone angles for 1, 2, 4 and 8 iterations, respectively. Brighter texels indicate a lower angle.

7 RESULTS

We have implemented the ray engine for tracing refracted eye rays, without the CPU-GPU load sharing scheme, and our algorithm (Figure 5), to compare their performance. Both algorithms also performed a refracted direction calculation for every intersection test.

7.1 Ray coherence

A basic assumption of our approach is that rays within a tile will remain more or less coherent after multiple refractions, and in most cases they still can be enclosed by tight cones. Images in Figure 4 shows how the opening angle of the cones increases with consequent refractions. Obviously, the cones are extremely accurate in the first few steps, and larger openings only appear near contours. Later on some cones expand as new refraction contours appear, but the main yield in this phase is that homogenous or already terminated areas have already been identified, and they are discarded without the CPU rearranging the remaining rays into a new array, as in the Ray Engine approach.

For the hierarchical ray engine implementation, we divided the 256×256 ray array into 16×16 tiles. The size of the tiles is somewhat arbitrary, but limited by the maximum size of point primitives the hardware can render. The optimum may be dependent on the complexity of the scene. For our test scenes, we found that further decreasing the tile size resulted in performance loss, meaning that the overhead of more processed vertices and larger textures became more significant than the gain from better coherence.

As depicted in Figure 1, the enclosing cones for the tiles were computed in a shader. The results shown in Figure 7 were obtained for different geometries, with the maximum path length set to 5, on an NV6800GT. A refractive surface was visible in every pixel. Considering that it takes at least two iterations to discard a path that has entered an object, a minimum estimate for the throughput can be given. With 256×256 pixels, 5000 triangles, 2 iterations, and an FPS of 0.23 our ray engine implementation computed $150M$ intersections per second. With the hierarchical approach achieving 3 FPS, this figure reaches $2G$. The improvement over the naive approach is very much dependent on the complexity of the geometry. If there are few primitives, the ray casting does not take much time, and the constant overhead of constructing the enclosing cones does not pay off. However, as the number of primitives increases, the situation is reversed. Already at a triangle count of 100, the new algorithm runs twice as fast. Furthermore, for 256×256 eye rays, all hitting a refractive surface, frame rates enough for real time rendering have been achieved. This makes it possible to interactively and accurately render visible refractive objects, or, when using the technique to trace photon paths, correct caustic patterns.

Figure 5: Images rendered using the hierarchical algorithm at 256×256 resolution.

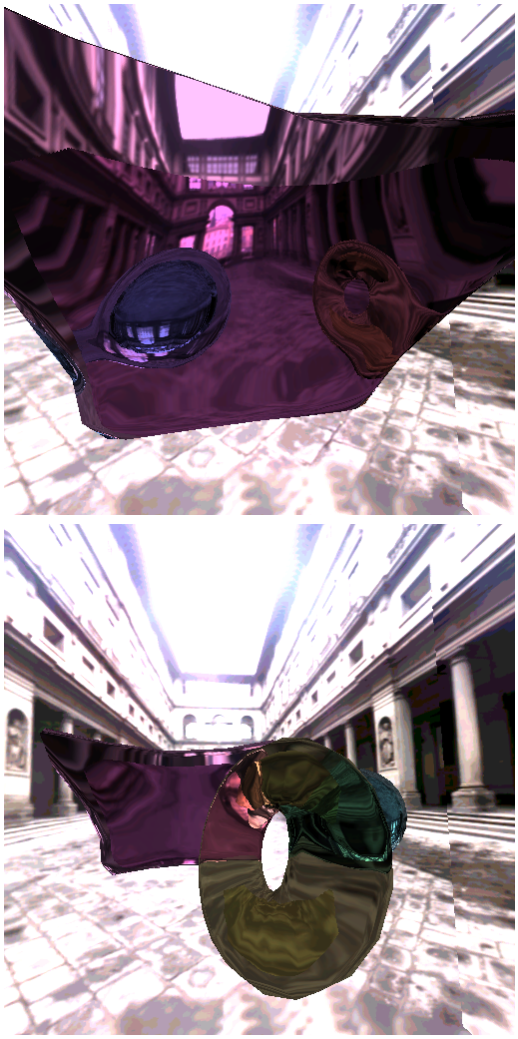


Figure 6: Images rendered using the hierarchical algorithm at 512×512 resolution.

8 COMPARISON WITH OTHER ALGORITHMS

Compared to the brute force algorithm, Foley and Sugerman[3] reported relative speedups for their grid and kd-tree GPU acceleration structures between a factor of 5 and 9. Their test scenes contained up to 100000 triangles. As complex acceleration structures have to be built using the CPU, the algorithms are very well suited for high triangle count static scenes, and impose no constraints on the coherence of the rays whatsoever.

With a quite different set of preferences, the hierarchical ray engine can achieve a relative speedup of 15, outperforming the spatial hierarchies. However, the algorithm is linear in the number of triangles, making it less suitable for scenes with more than several thousand triangles, and it also assumes strongly coherent rays. However, as stated in the introduction, the problem of caustics and visible multiple refractions, the only applications of ray tracing where there is no reasonable incremental alternative, require these features exactly.

The approximate ray tracing approach[7] has a very similar area of applications compared to our algorithm. Its main limitations are that it uses pre-computed distance impostor textures, and only two refractions can be handled. It is impossible to render a refractive object seen through a refractive object. Furthermore, the refraction computation is only accurate for a single star-shaped object, the geometry of which can be captured in a single cube map. Therefore, while the approxi-

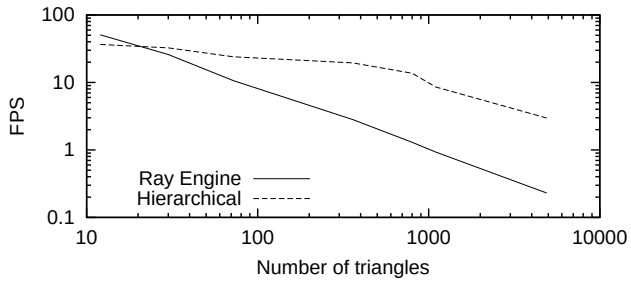


Figure 7: Frames per second rates achieved for rendering refractive objects.

mate ray tracing is extremely fast and convincing for some spacial cases, our algorithm offers accurate multiple refractions and a potential to handle dynamic objects or even liquids.

9 FUTURE WORK

The current implementation is only using the algorithm to query an environment map for rays that do not hit any refractive surface. Although it is theoretically straightforward, we still need to work on an implementation that demonstrates the method's applicability for tracing scenes with non-ideal objects, and, more importantly, for generating caustics.

REFERENCES

[1] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In *An Introduction to Ray Tracing*, pages 201–262. 1989.

[2] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In *Proc. Graph. Hw. 2002*, pages 1–10, 2002.

[3] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22, New York, NY, USA, 2005. ACM Press.

[4] V. Havran. *Heuristic Ray Shooting Algorithms*. Czech Tech. Univ., Ph.D. dissertation, 2001.

[5] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 101–108, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[6] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. on Graph.*, 21(3):703–712, 2002.

[7] László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate ray-tracing on the gpu with distance impostors. In *Computer Graphics Forum, Proceedings of Eurographics 2005*, volume 24, Dublin, Ireland, 2005. Eurographics, Blackwell.

[8] I. Wald, C. Benthin, P. Slussalek, and M. Wagner. Interactive rendering with coherent ray tracing. In *Eurographics '01*, 2001.

[9] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In *Rendering techniques '99*, volume 10, pages 235–246, Jun 1999.

Improved Illumination Estimation for Photon Maps in Architectural Scenes

Robert F. Tobler
VRVis Research Center
Donau-City Str. 1/3
1120 Wien, Austria
rft@vrvis.at

Stefan Maierhofer
VRVis Research Center
Donau-City Str. 1/3
1120 Wien, Austria
sm@vrvis.at

ABSTRACT

The photon map algorithm provides a number of advantages for fast global illumination algorithms. In order to calculate the illumination at each point in a scene, the photon density needs to be estimated. Standard estimation methods will have problems with typical architectural scenes, as walls and corners will lead to undesirable light and shadow leaks.

We introduce a new method for estimating photon density in photon maps, that is especially suited for calculating global illumination in architectural scenes. By providing additional information using a limited number of ray-casts, shadow and light leaks can be significantly reduced, thereby resulting in a significant improvement in the speed and accuracy of global illumination algorithms based on photon maps.

Keywords

Photon mapping, global illumination, density estimation, bias.

1. INTRODUCTION

The standard way of calculating global illumination for architectural scenes, is to simulate the propagation of photons using some ray tracing algorithm. If no information about these rays is stored, and all illumination estimates are independent of each other, no bias is introduced.

In order to speed up the algorithms, information about the rays can be stored. Although this introduces a bias, the resulting algorithms such as irradiance caching [War88, War92] and photon mapping [Jen96] are significantly faster than unbiased methods.

A number of improvements to the original photon map algorithm have been published, that try to reduce the inherent bias. Hey and Purgathofer use an average of several oriented photon maps [Hey01], Lavignotte and Paulin extend the object boundaries for photon storage in polygonal scenes [Lav02]. Other improvements of photon mapping are the use of density control [Suy00, Pet98], and the use of a convex hull [Jen01, Jen02] and the use of a ray-cache based on a set of spheres [Las02]. A very recent approach to reduce the bias is the extension of the photon map to include complete rays [Hav05].

In this paper we will demonstrate techniques that can be used to improve the density estimation for algorithms based on standard photon maps that only store photon locations.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

2. THE PHOTON MAP ALGORITHM

The basic algorithm of the photon map is based on the simulation of the propagation of photons starting at the light sources. Each interaction of a photon with a surface is recorded in a data structure that allows fast searches for nearby hits, e.g. a k-d tree. Once enough photons have been simulated, illumination at each point in the scene can be approximated by retrieving the closest photon hits at the query point. Using the k-d tree the n closest hits are retrieved, and due to the resulting distance of the n-th closest photon hit a projected surface area can be computed (see figure 1). Dividing the power of the n closest hits by this projected surface area, results in an estimate for the radiance at the query point.

Changing the metric for retrieving the n closest photons

Due to the nature of the algorithm, the photon hit positions in the k-d tree approximate the surfaces of the geometry used in the simulation. Nevertheless a standard query of the n-th closest photon hits will retrieve photons that may lie on different surfaces. In order to reduce the error in the computation of the projected area, Jensen [Jen02] suggested using a modified metric for computing the n closest photon locations, by stretching space in the direction of the surface normal of the query point. With this modified metric the n-th closest photons around the query point form an ellipsoid that is squashed and approximates the surface near the query point. This improvement reduces the error of including photons from different surfaces in the photon query.

Weighted reconstruction kernel

Another improvement published by Jensen [Jen02] is the use of a weighted reconstruction kernel. Instead of equally weighting all photon hits in the vicinity of the query point, a reconstruction kernel is placed around the query point, emphasizing photons that are close to the query point and deemphasizing photons that are further away from the query point. Typically a cone filter is used.

Separating global illumination and specular effects

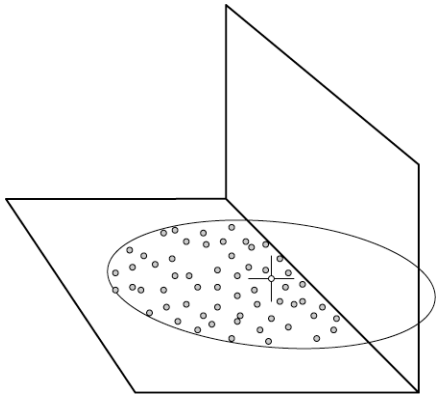
In order to compute the caustic effects produced by highly specular reflection and refraction, Jensen proposed to separate the simulation of photons reflected at highly specular surfaces from standard global illumination computation: global illumination calculation can be performed with less photons, if effects with high spatial frequency need not be computed, whereas the computation of caustic illumination effects needs high photon density for accurate simulation. Thus a low number of photons designated for global illumination are simulated, and a separate simulation with photons directed at highly specular surfaces is used for computing caustics.

3. PROBLEMS WITH PHOTON MAPS

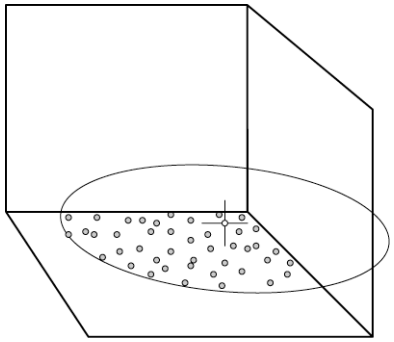
As already mentioned, the photon map constitutes a biased global illumination algorithm. The reason for the bias is the necessity to estimate illumination based on a finite sized kernel encompassing the n closest photons around a query point. This estimation can lead to various problems, the most visible of them being light- and shadow leaks. Especially in architectural scenes, walls in a building can exhibit both these problems.

Shadow Leaks

In a number of geometric configurations that are quite common in architectural scenes, the described way of estimating photon density will lead to an overestimation of the area covered by the photons that have been retrieved. This leads to shadow leaks , a special case of boundary bias, close to the edges and corners of rooms. Figure 1 shows two examples of such geometric configurations.



(a) Illumination estimate near wall



(b) Illumination estimate near corner

Figure 1. Geometric configurations leading to shadow leaks: note that photons on the vertical faces of the walls are not shown, as they do not contribute to the projected area. Photon hits are marked by small circles, the query point is marked by the cross-hair.

Light leaks

Another type of artifacts, namely a so-called occlusion bias, of the photon map algorithm happens in geometric configurations where photons in the vicinity of the query point should not contribute to the illumination, but are included as they are the closest photons. This typically happens close to walls, and leads to unwanted light leaks. Figure 2 demonstrates a geometric configuration leading to such a light leak.

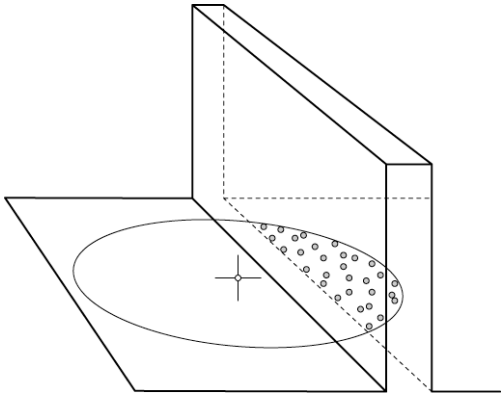


Figure 2. A geometric configuration leading to a light leak.

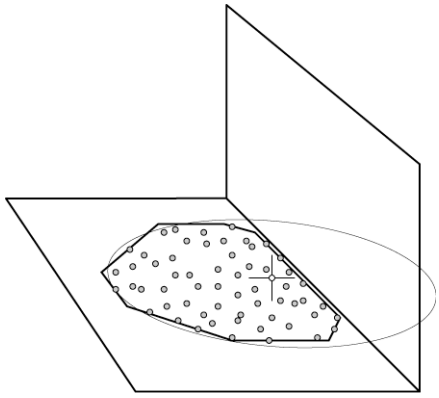


Figure 3. Improved estimate of projected area with OctoBoxes.

4. IMPROVED DENSITY ESTIMATION FOR PHOTON MAPS

In order to improve the density estimation for a query point it is necessary to compute a better estimate of the area covered by the n closest photon hits. These improved area computations should not slow down the query process, while giving a significantly improved estimate of the actual area.

Based on this requirement, some variant of 2D bounding boxes around the closest photon hits seems to be a promising approach. However standard bounding boxes overestimate the covered area in the case of constant photon density: the query method for calculating the n closest photons will return a nearly circular set of photons, and the bounding box of these photons will be about 27% larger than the actual area (area ratio of the 2D bounding box to the disk of retrieved photons).

8-sided 2D Bounding Boxes: OctoBoxes

In order to improve the estimate we use extended bounding boxes, that maintain additional extreme values in the four median directions (45º, 135º, 225º, and 315º). These 8-sided 2D bounding boxes, we call OctoBoxes, can be implemented to work nearly as fast as the standard bounding boxes, and will result in a significantly improved estimate of the area estimate of the n closest photons. This is a fast and simplified implementation of the convex hull idea presented by Jensen [Jen01, Jen02].

With this modification, the overestimation of the area in the limit case of a constant photon density is reduced to 5.5% (area ratio of the enclosing octagon to the disk of retrieved photons). Figure 3 shows an example of the improved estimate of the projected area possible with such OctoBoxes.

The use of OctoBoxes will eliminate shadow leaks close to the edges and corners of architectural scenes and thereby improve the illumination estimates of the photon mapping algorithm significantly. However there are still configurations that lead to unwanted shadow leaks. Figure 4 shows an example of a geometric configuration that leads to a shadow leak, even if OctoBoxes are used for computing the projected area of the n closest photons.

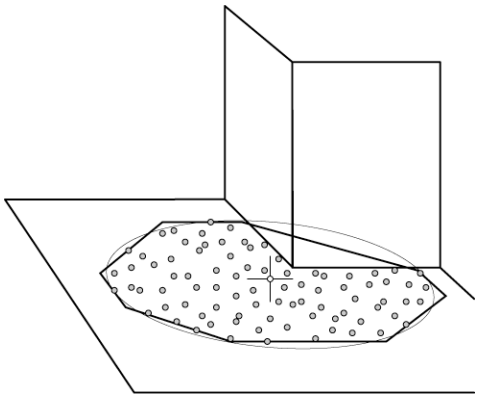


Figure 4. Geometric configurations leading to shadow leaks even if OctoBoxes are used.

Per Octant OctoBoxes

The problem with such configurations is, that the set of the n closest photons is not convex anymore. In order to overcome this problem, we use more than one OctoBox: based on the query point we partition the tangent plane into its 8 octants. For each of these octants we maintain a separate OctoBox for estimating the area of those of the n closest photons that fall within the respective octant. Thereby, we can calculate an improved estimate for the area of the n closest photons, for a number of cases in which these photons cover a non-convex area. Two examples of geometric configurations that can be handled by this improved heuristic can be seen in figure 5.

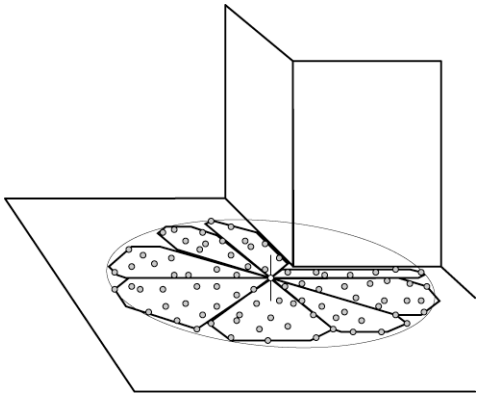


Figure 5. An example geometric configuration that can be handled with 8 OctoBoxes.

The use of a separate OctoBox for each octant will improve the area estimate in a number of cases, but as it is a simple heuristic, there are more complex illumination cases where this method will not generate an adequate result.

In cases of walls or corners at an angle between the octant angles (e.g. at 22.5º) the OctoBoxes will lead to their maximal possible error: in extreme cases, e.g. a very thin illuminated ribbon at an angle of 22.5º, the error can get arbitrarily large, however these cases are very rare, so in most situations the heuristics work pretty well.

If any of the octants receive a very small number of photons, the resulting estimate can be statistically unstable. In order to avoid such cases, we interpolate between the simpler heuristic (only one OctoBox) and the more sophisticated heuristic (one OctoBox per octant), based on the number of photons. If each octant receives enough photons (more than about 50) we use the sophisticated heuristic exclusively.

Note that the use of these OctoBoxes does not significantly slow down the algorithm.

Although these techniques based on OctoBoxes result in a significant improvement by eliminating most instances of shadow leaks in architectural scenes, they cannot eliminate light leaks, as no additional information about the geometric configuration around the query point is obtained.

Geometry Feelers

In order to overcome the problem of light leaks, a number of purely statistical methods have been tested, that evaluate the distribution of the n closest photons. However none of these statistical methods were stable enough to eliminate typical light leaks in architectural scenes. For this reason a different approach was chosen. By shooting so-called geometry feelers, i.e. rays that return the distance to the closest object, the geometry in the vicinity of the query point can be investigated. By shooting rays parallel to the plane of the projected area of the illumination query, that start from the query point and run in four or eight major directions around the query point, an estimate about the free space around the query point can be made (see figure 6).

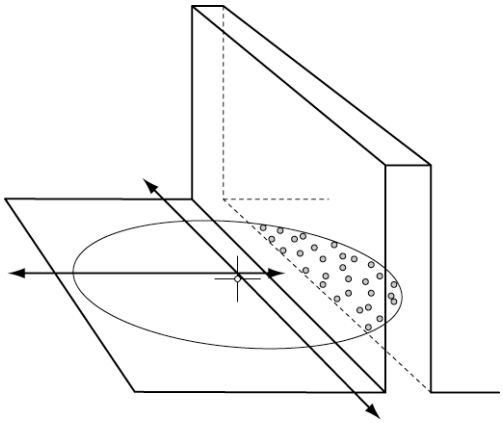


Figure 6. Geometry Feelers for estimating free space around the query point.

The geometry feelers are then used, to create a bounding box or OctoBox around the query point that is used as a filter for all photons: only photons inside this box are used for illumination estimates.

We tested this heuristic with a number of architectural scenes, and found out, that 4 geometry feelers can still lead to visible artifacts, whereas 8 geometry feelers per query point will eliminate most light leaks in architectural scenes.

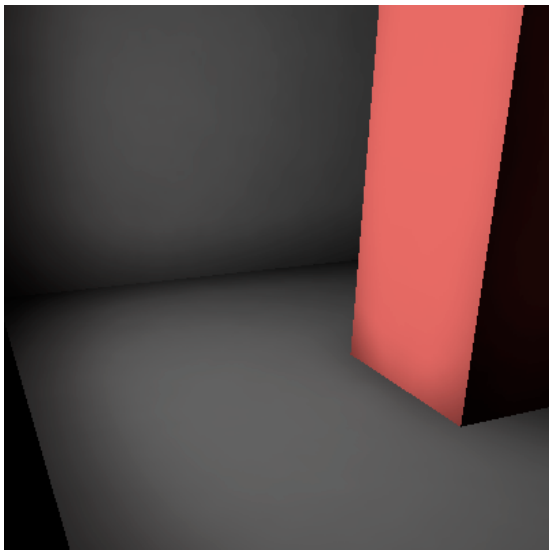
The use of the geometry feelers does slow down the computation of the photon map somewhat. Note that the actual reduction in computation speed is dependent on the relative performance of the k-d tree query with respect to the ray-casting implementation. In typical examples the observed slow-down was between 15% and 40%.

Note that the technique is not guaranteed to eliminate all light leaks. It is possible to find geometric configurations, where the geometry feelers do not adequately represent the surrounding geometry, e.g. when the geometry feelers exactly go through small holes in otherwise solid walls. Nevertheless these configurations are very unlikely in typical architectural scenes, and thus the heuristic works quite well.

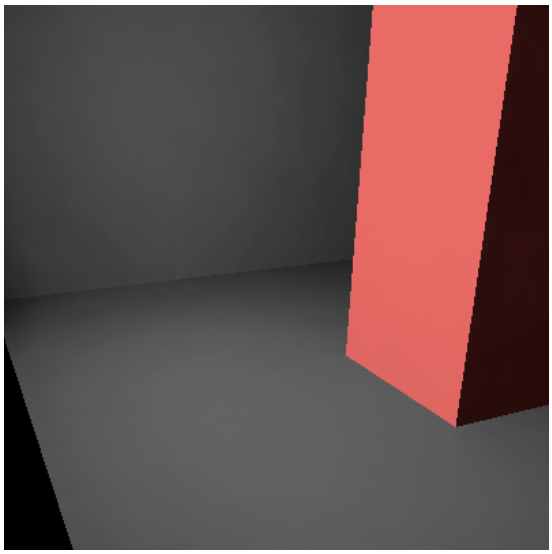
5. RESULTS

All the introduced heuristics have been built into a global illumination system, that precalculates illumination for interactive walkthroughs. The illumination is stored as light maps that are maintained in parallel to the texture maps for each object, and rendered using multiple texturing. The following examples show the improvements that have been obtained with the heuristics.

The first example shows the effect of the use of the OctoBoxes in order to fix shadow leaks (figures 7).



(a) shadow leaks



(b) no shadow leaks with OctoBoxes

Figure 7. Fixing shadow leaks with the use of OctoBoxes.

The second example shows the effect of the geometry feelers for fixing light leaks (figure 8).

(a) light leaks

(b) no light leaks with geometry feelers

Figure 8. Fixing light leaks with the use of geometry feelers.

Finally in the last examples (figure 9) we show a global illumination solution for a typical architectural scene, calculated using the presented improved heuristics.

Figure 9. Examples of global illumination in architectural scenes calculated using the presented heuristics.

6. CONCLUSION

We presented two heuristics that significantly increase the quality of photon map illumination estimation, by improving the area estimate of the projected area of the n closest photons, and preventing the propagation of illumination through geometric obstacles.

Although these heuristics do not result in a complete removal of the problematic light and shadow-leaks, and they are geared towards scenes with architectural characteristics such as walls and corners, it has been shown that they work well for architectural scenes.

7. REFERENCES

[Hav05] Havran, V., Bittner, J., Herzog, R., and Seidel, H.-P. Ray Maps for Global Illumination in Rendering Techniques 2005, July 2005.

[Hey01] Hey, H., Purgathofer, W., Global Illumination with Photon Mapping Compensation. Tech. Rep. TR-186-2-01-04. Vienna University of Technology, January 2001.

[Jen96] Jensen, H. W. Global Illumination using Photon Maps in Rendering Tecniques '96, pp.21-30, June 1996.

[Jen01] Jensen, H. W. Realistic Image Synthesis using Photon Mapping. A. K. Peters, Ltd., 2001.

[Jen02] Jensen, H. W. A practical Guide to Global Illumination Using Photon Mapping in Siggraph 2002 Course Notes CD-ROM, Course 43, July 2002.

[Las02] Lastra, M., Urena, C, Revelles, C, Montes, R. A Particle-Path-based Method for Monte-Carlo Density Estimation. Poster Paper Proceedings of the 13th Eurographics Workshop on Rendering, pp. 33-40, June 2002.

[Lav02] Lavignotte, F., Paulin, M. A New Approach of Density Estimation for Global Illumination. In Proceedings of WSCG 2002, pp. 263-270, 2002.

[Pet98] Peter, I., Pietrek, G., Importance Driven Construction of Photon Maps. In Rendering Techniques '98, Drettakis G., Max, N., (Eds.), pp. 269-280, 1998.

[Suy00] Suykens, F., Willems, Y. D. Density Control for Photon Maps. In Rendering Techniques 2000, Peroche B., Rushmeier H., (Eds.), pp. 23-34, June 2000.

[War88] Ward, G. J., Rubinstein, F. M., Clear, R. D. A Ray Tracing Solution for Diffuse Interreflection. In SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, New York, USA, vol.22, ACM Press, pp. 85-92, Aug 1988.

[War92] Ward, G. J., Heckbert, P. Irradiance Gradients in Proceedings of the Third Eurographics Workshop on Rendering, Course pp. 85-98, May 1992.

Interpretation of Overtracing Freehand Sketching for Geometric Shapes

Day Chyi Ku
School of Engineering and Design
Brunel University
Uxbridge
UB8 3PH, UK
Daychyi.Ku@brunel.ac.uk

Sheng-Feng Qin
School of Engineering and Design
Brunel University
Uxbridge
UB8 3PH, UK
Sheng.Feng.Qin@brunel.ac.uk

David K. Wright
School of Engineering and Design
Brunel University
Uxbridge
UB8 3PH, UK
David.Wright@Brunel.ac.uk

ABSTRACT

This paper presents a novel method for interpreting overtracing freehand sketch. The overtracing strokes are interpreted as sketch content and are used to generate 2D geometric primitives. The approach consists of four stages: stroke classification, strokes grouping and fitting, 2D tidy-up with endpoint clustering and parallelism correction, and in-context interpretation. Strokes are first classified into lines and curves by a linearity test. It is followed by an innovative strokes grouping process that handles lines and curves separately. The grouped strokes are fitted with 2D geometry and further tidied-up with endpoint clustering and parallelism correction. Finally, the in-context interpretation is applied to detect incorrect stroke interpretation based on geometry constraints and to suggest a most plausible correction based on the overall sketch context. The interpretation ensures sketched strokes to be interpreted into meaningful output. The interface overcomes the limitation where only a single line drawing can be sketched out as in most existing sketching programs, meanwhile is more intuitive to the user.

Keywords

Freehand sketching, multistroke sketching, calligraphic interface, design

1. INTRODUCTION

Sketch interface is an important, natural application to support conceptual design. A sketch system implemented in a computer has the advantage whereby further manipulations and processing, such as 3D modelling from 2D sketch, can be made directly with minimal steps within a short time. However, this is realistic only if the system allows the flexibility of transferring ideas directly from designers through a series of freehand sketch. To support this process, the system should provide an interface that is natural to the user as sketching with pen and paper. Users may find sketching with extensive menus difficult as a result of frequent interruptions.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

On the other hand, the system will also need to be able to correctly interpret the user's intent from the drawing of the sketch. A trade-off is often required in the design of the calligraphic interface between being natural, easy-to-use, and that of accuracy of the system's interpretation of user's intent.

One problem in the design of a natural but accurate calligraphic interface is that of interpreting overtracing of freehand sketch. Overtracing is frequently used to enhance and complete an edge during freehand sketching. A system that supports overtracing, i.e., accepts multiple stroke inputs, has the advantage of providing more drawing freedom to the user. However, at the same time, overtracing further increases the number of possible interpretations of a sketch, and as such, making the system more susceptible to making mistakes in interpreting the user's actual intent.

In this paper, we are concerned specifically with the interpretation of overtracing freehand sketches of geometric objects. Although there are sketch systems that support overtracing inputs, they do not actually interpret them [FIO02a, GRO00a, KAR05a], or provide limited interpretations (e.g., only supports certain sketching primitives or has a very strict

definition of how a sketch is interpreted) and as such, not applicable to general cases [MIT02a, SHE04a].

Here, we developed a system that supports and interprets overtracing freehand sketch of general geometric objects (Figure 1). The work presented in this paper is a part of our on-going project that is aimed at reconstructing 3D models from 2D sketches. The preliminary system is developed to address the problem of interpreting overtraced strokes, i.e., multiple strokes that are part of the same geometric primitives.

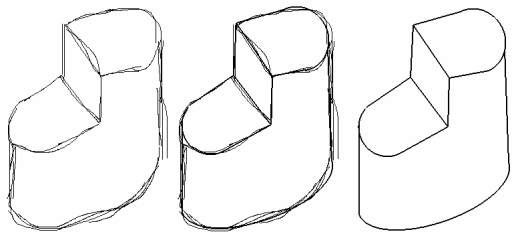


Figure 1: A freehand sketch with overtracing and its tidy-up output from our system.

There are two important differences between our system and other earlier, calligraphic interface systems such as in [MIT02a, SHE04a]. First, our system supports and interprets both straight-line and curve overtracing sketches. In [SHE04a], the system can only interpret straight-line input. Second, our system interprets overtracing strokes with various curvatures and represents them in parametric equations that best represent the strokes. In [MIT02a], the overtracing strokes are represented by polylines after grouping into core curves and only curves with low curvatures are showed in their examples.

Furthermore, some of the systems impose frequent interruptions to users during the sketching process. This is normally associated with interactive systems that prompt users for selection of choices [IGA97a]. The frequent interruptions can be a source of distractions that can impede the flow of thoughts of the designer, and as such, should be addressed in the design of a calligraphic interface.

Therefore, the interpretation process in our system is carried out automatically and designed to have minimum interruption to the user. To achieve this, we designed the system to carry out the interpretation process between sketching sessions rather than interrupting the user during sketching for further inputs to clarify ambiguous cases. This allows a user to draw in a more intuitive and natural way without diverting attention from the sketching flow.

2. RELATED WORKS

Computer calligraphic interface has been receiving more attention over recent years. However, despite the availability of hardware such as the stylus tablet, there are still many problems left to be solved in the development of a full-fledged, functional free-hand sketching calligraphic interface, such as interpretation of multiple strokes, in-context classification and clustering of strokes, inferring constraints from freehand sketches for tidy-up and beautification.

Perhaps Pavlidis [PAV85a] made the first attempt to infer constraints from initial freehand sketch to automatically tidy-up rectilinear drawings. However, the method is bound to produce unintended and undesirable results in practice and negative constraints were suggested to address the problem [PAV85a]. Similarly in Easel [JEN92a], Jenkins and Martin developed a system that can automatically analyze and tidy-up sketches. Easel introduced more constraints and processed freeform curves [JEN92a]. Pegasus [IGA98a, IGA97a] is a prototype system for beautification of freehand sketching through an interactive process. The system generates several candidate strokes based on geometry constraints for user selection. This interaction process is to ensure that the resulting sketch is as closely desired by the user, i.e., the precise and correct sketch that the user had in mind but unable to accurately draw himself. However, the process of selecting candidates, stroke-by-stroke, is somewhat distracting to the user. Consequently, the system is not suitable for conceptual design sketching, where the designer must be allowed to sketch without interruption to ensure a continuous flow of ideas.

In addition to the tidy-up process, there are other research efforts focused on sketch interpretation problems such as the strokes classification and clustering. For example, Shpitalni and Lipson [SHP97a] presented a method for classifying input strokes in an online sketching system that is based on linear least squares fitting to a conic section equation. They also introduced new endpoints clustering scheme based on adaptive tolerances [SHP97a]. Qin [QIN00a] introduced a system that classified strokes using adaptive threshold and fuzzy knowledge with respect to curve's linearity and convexity. After that, 2D primitives are identified and a 2D relationship inference engine is used to study their relationship for 3D recognition.

Although advances have been made in developing a functional calligraphic interface for freehand sketching, in general, most of the current 2D freehand sketch interpreters have the limitation of either not supporting overtracing sketches [JEN92a, IGA98a, IGA97a, PAV85a, QIN00a, QIN00b, SHP97a], or have limited support ([SHE4a], [MIT02a]). SMARTPAPER [SHE4a] groups overtracing strokes into segments. The grouping

process was carried out in two passes where the distance between end points and the slopes of the strokes are checked against each other. However, SMARTPAPER is limited to straight-line input with limited configurations only. Furthermore the system does not support curves input. There is no discussion of wrongly drawn lines correction and hence it is assumed that the system is limited to interpret sketches with correct strokes. 3D SKETCH [MIT02a] supports both overtracing and hatching sketch. Strokes are divided into core strokes (strokes that touch the characteristic curves of the object), and hatching strokes (strokes that are mapped to the faces of the object). Although the system can interpret overtracing strokes (e.g., by grouping strokes into bundles), there is no explanation of how the characteristic strokes are distinguished from the hatching strokes. Furthermore, the system used polylines to represent the core curve limited its accuracy in representing curves with higher curvature.

There are other sketch systems that support overtracing strokes but do not interpret them as part of the sketch. In [GRO00a], the system filters out overtraced lines to produce simpler, approximated drawings (e.g., filtering out elements that are smaller than a specified size) rather than performing interpretations in the context of the sketch as a whole. In [KAR05a], the sketch recognizer allows overtracing symbol recognition based on image processing techniques, e.g., it relies entirely on the symbol libraries for the recognition where the sketch is examined in pixel and hence no work is done on interpretation of individual stroke. In [FIO02a], overtracing is used to edit an existing stroke that is represented in cubic Bezier splines. The movements of oversketching is sampled and interpreted specifically as transformation attractor to the underlying curve's control points. The overtracing strokes are used as gesture input to alter the existing sketch rather than sketching information that actually add more lines to the existing sketch.

In this paper, we propose an interactive calligraphic interface that addresses the limitations discussed above. Referring to the systems discussed above, we use Qin's curve classification, Shiptalni and Lipson's conic fitting equation, and others geometry constraints as the backbone of our system to interpret and tidy-up freehand sketch. However, stroke pre-processing, i.e., segmentation, is not covered in this paper. All input strokes are taken to represent only a segment or part of a segment. However, the system here can be easily extended to include pre-processing algorithms for segmentation such as those described in [QIN00b, SEZ01a].

3. SYSTEM OVERVIEW

We propose the interpretation of overtracing strokes by carrying out a set of tests automatically to group and tidy up the sketched strokes into segments. The output of the system is edge-vertex graph with initial sketched strokes associated to the edges. The graph can be transferred into CAD or 3D modelling systems for further processing and manipulation while the edges can be reproduced in sketchy style with the sketched strokes information. Another advantage to our proposed system is that the new edges that looked similar to the original sketching style can be produced from the information obtained during the sketching process. However, for this paper, we focus only on planar objects and objects with ellipsoidal curves for inputs to reduce the system's bounds in producing unintended and undesirable result. Additional work is needed to extend the system to support the interpretation B-spline curves and freeform objects.

Interruptions such as prompts for clarifying ambiguous strokes are made at the end of the drawing session. For example, our system prompts the user to decide whether an ambiguous stroke should be deleted or kept with the existing drawing only at the end of the drawing session.

The system can take in the sketch directly from the user through a tablet with a digital pen or mouse. A stroke is a set of points that is captured during the period when the mouse button is pressed down, moved, and released. An edge refers to the intersection of two faces of a solid object, which in 2D drawing is represented by strokes. In many drawings an edge is often composed of a set of discontinuous strokes. The overtracing strokes are used to complete, correct or enhance an edge in the sketch. The strokes will be processed by the 2D sketch interpreter and automatically grouped together to represent the associated edges.

The 2D sketch interpretation is divided into five stages: grouping and fitting of strokes, end points clustering, parallelism correction, in-context interpretation and user interactive selection. First, all classified strokes (using the linearity test) will be grouped according to their positional relations. After that, the grouped strokes are fitted collectively into an appropriate parametric equation in the least square sense. A tidied-up sketch will be produced from the equations. Lastly, lines with similar gradients (within some tolerance levels) are adjusted to be parallel to each other, thus reflecting the user intention because it is not possible for the user to draw exactly parallel edges in freehand sketch. When there are strokes that cannot be interpreted correctly according to geometry constraints, the system will tag the strokes as ambiguous cases. The system only prompts the

user for further action and correction of the ambiguous cases at the end of the drawing session.

After the interpretation process, the system is then able to generate tidied-up 2D single line drawing, while at the same time, still have initial sketchy strokes linked to the appropriate tidied-up edges.

4. CLASSIFITION AND GROUPING

4.1 Stroke Classification

All drawn strokes will be classified into either one of the categories, straight-lines or curves. The linearity test is used to classify a straight-line from a curve [QIN00a]. It is simpler and faster compared to the conic curve equation used in [SHP97a]. This is because the linearity test involves only a simple calculation of two parameters, while the conic curve equation requires the calculation of five parameters in the quadratic equation.

The linearity is defined as the ratio of the distance between two endpoints of a stroke to the accumulative chord length between sequences of points captured.

The linearity value is a floating point between zero to one. A greater linearity value for a stroke indicates that the stroke is more likely to be a straight-line rather than a curve. An ideal straight-line will have unity linearity, which rarely happened in freehand sketching. Therefore, we set an arbitrary straight-line tolerance for the classification.

Each classified stroke will be fitted either into a general straight-line equation to generate a straight-line, or a general conic curve equation to generate a curve. With equations, strokes can be represented in a more general and effective way compared to using a list of points. Furthermore, the representation of input strokes with parameterised equations allow for more effective and efficient tests of input strokes during the interpretation process.

4.2 Strokes Grouping and Fitting

The strokes grouping process groups sketched strokes into the appropriate edges that they represent. However, the grouping process only tries to group strokes in the same category, i.e., lines only grouped with lines, and curves only grouped with curves. The grouped strokes are fitted with equations to obtain parameters that best represent them.

4.2.1 Line Segment Grouping and Fitting

A straight-line is approximated by fitting data points of the strokes into a line equation that give the minimum least-square error [ONE83a]. The fitted line end points are the points on the equation that have the minimum Euclidian distance from the sketched stroke end points. All straight-lines will be tested to determine whether they should be grouped together to represent the same edge. The line segments will be grouped together if they are close in position and orientation to each other.

4.2.1.1 Distance Tests

A series of distance tests is used to determine the smallest distance between two line segments. The tests are carried out in sequence as follows:

1) Get the smallest Euclidean distances between endpoints of lines *A* and *B* (Figure 2). The resulting distance is compared with a threshold value. The threshold value represents the largest tolerance of the distance between two line segments to be grouped together. The threshold value varies according to the lines' length, as in [SHP97a]. If the two line segments (*A* and *B*) pass the threshold test, go to the angular test, otherwise go to step 2.
2) Calculate the perpendicular Euclidean distances from line endpoints of one line to the other line segment, as shown in Figure 2. Eliminate the distances that corresponding projection points are out of the other line segment, e.g., (a) and (d) in Figure 2. Compare the smallest distance among the remains, (b), with a threshold value. If the distance is smaller than the threshold, go to the angular test. Otherwise the two line segments are grouped into separate edges.

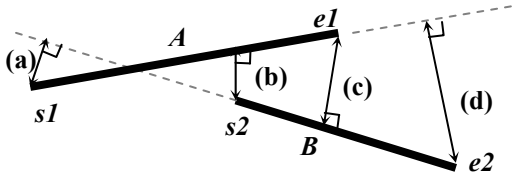


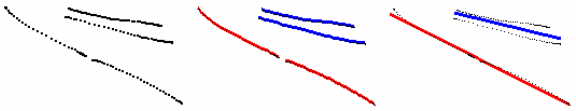
Figure 2: Distance calculated between *A* and *B*.

4.2.1.2 Angular Test

The Angular test is to ensure that strokes grouped as one edge are pointing in the same direction.

The absolute dot product value, which is the smallest angle between the lines *A* and *B*, is calculated. The two lines will be grouped into one segment if the angle is smaller than a threshold value. The threshold value is adjusted according to the longest length of the lines *A* and *B*. The value is calculated to be inversely proportional to the length of the corresponding line, e.g. the longer line will have a smaller threshold value. This is because the longer length of lines will "look" more apart than shorter line with the same angle.

The lines do not have to intersect, or overlap each other to be grouped together. The grouped strokes will undergo another round of straight-line least-square fitting to update its equation and endpoints (Figure 3).



(a) (b) (c)

Figure 3: Line Grouping: (a) Initial sketch; (b) Grouped sketch; (c) Fitted result.

We assume that earlier drawn stroke is more likely to represent an edge position and orientation, while strokes drawn later are the overtracing to enhance or complete the edge. Therefore, earlier drawn stroke is used as reference to test against candidate strokes to determine if the strokes should be grouped into the edge. If a candidate stroke fails the tests for all existing edges, then it will be grouped as a new edge and used as reference for the edge.

4.3 Curve Grouping and Conic Fitting

Strokes classified as curves by the linearity test will undergo the curve fitting process before curve grouping can be carried out. They will be fitted using a conic curve equation [SHP97a] to obtain the curve parametric equation. The fitted result will fall in one of the following category: straight-line, parabola, hyperbola, or ellipse.

However, straight-lines will not be generated due to the pre-processing of strokes through the linearity test. Hyperbola and parabola are considered as special cases by our system on the assumption that they rarely occurred in sketched geometric objects. Consequently, we only consider the elliptical curves that result from the fitting. Circle is treated as special case of ellipse where the major and minor radii are identical. After that, all sketched curves are represented by the parametric equations generated from the conic fitting procedure.

A general conic curve can be described by the following equation [BOW83a]:

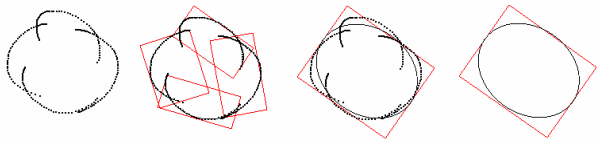
$$Q(x,y) = ax^2 + 2hxy + by^2 + 2gx + 2fy + c = 0$$

In our system, we need to find the least square fitting based on the distance between the captured sketching points and the equation. The fitting problem is then reduced to minimizing the following function:

$$E = \sum_{i=1}^{n} \left(ax_i^2 + 2hx_i y_i + by_i^2 + 2gx_i + 2fy_i + 1\right)^2$$

We obtain the coefficients (*a, h, b, g,* and *f*) by solving the partial derivatives of *E* equals to zero. The central point, major and minor radii, and the rotation angle of an ellipse can be obtained from the equation as in [QIN99a].

A bounding box, the smallest rectangle to enclose a fitted curve, is used to test for the curves adjacency for grouping them together. If the bounding boxes overlap one another, the sketched strokes associated with the curves will be grouped together. The grouped strokes will be fitted again. The overlapping test will be repeated for new fitted curve until there is no more overlapping bounding boxes in the sketch (Figure 4).



(a) (b) (c) (d)

Figure 4: Curve Grouping and Fitting: (a) Initial sketch; (b) Generate bounding box for each stroke; (c-d) Grouped strokes with fitted result.

4.3.1 Curve Range

A curve range is calculated from its starting point to the ending point, with reference to the centre point of the curve [JEN92a]. The user has the freedom in sketching a curve in any desired direction. However, our system standardized all curve range in anti-clockwise. The ending angle can be a value greater than 360 degree, as shown in Figure 5.

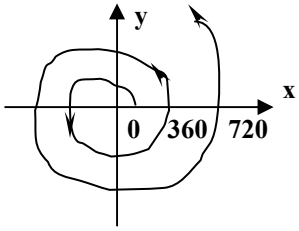


Figure 5: Curve range calculation.

Over-sketched and incomplete curves often occur in freehand sketching. Our system will tidy-up the curves into smooth ellipse or circle as shown in Figure 6. The curve range for grouped curves is updated automatically by our system, as shown in Figure 7.

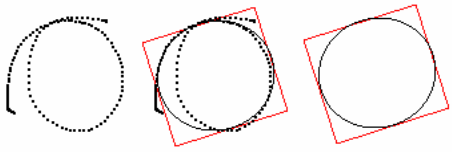


Figure 6(a): Over-sketched Curve.

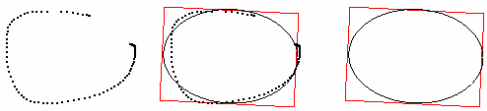


Figure 6(b): Incomplete Curve.

(a) (b) (c) (d) (e) (f)

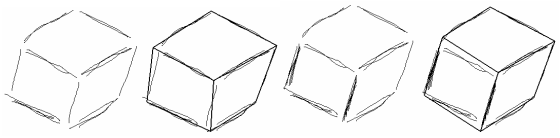
Figure 7: New curve range is determined from grouped curves: (a) Initial sketch; (b-c) Curve with initial range; (d) Additional stroke; (e) Strokes are grouped as a curve; (f) Curve with new range.

5. ENDPOINT CLUSTERING

After the grouping process, strokes representing a particular edge are approximated by a single line or ellipse parametric equation. The endpoint clustering process ensures that the corresponding edge endpoints meet together and that a close loop can be formed for edge-graph extraction.

5.1 Straight-line Junction Clustering

The straight-line junction clustering is applied for the line-to-line clustering. Before the clustering can be applied, the system first finds out endpoints of edges that are adjacent to each other and within the tolerance zone as discussed in [SHP97a]. Edges with endpoints that lie within the zone will be clustered together to form a junction. In case of a junction with more than two edges, the system will select two edges with the most number of strokes to determine the junction point, the rest of the edges will automatically snapped to the point (Figure 8).

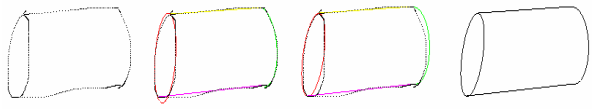


(a) (b) (c) (d)

Figure 8: Straight-line junction clustering: (a) Initial sketch; (b) Clustering result; (c) Modified sketch; (d) Updated clustering result.

5.2 Lines and Curve Junction Clustering

The curve edge will be adjusted based on the line edges endpoints position so that the curve edge endpoints (open curve) or boundaries (closed curve) will meet the line edges endpoints to form a close loop (Figure 9).

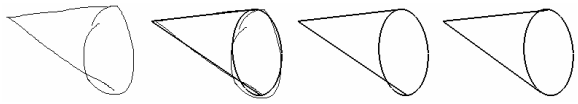


(a) (b) (c) (d)

Figure 9: Endpoint clustering for lines and curves junctions: (a) Initial sketch; (b) Fitted result; (c-d) Clustered result.

The orientation of the curve is determined to be parallel to the imaginary line segment L1 that connects the endpoints of lines that are to form junctions with the curve. After that, the curve's central point is determined to be the middle point of the line segment L1. The major or minor radius of the curve is calculated based on the length of L1, with the other radius being adjusted according to the distance from sketched curve to central point. For open curve, new curve range is determined based on the line endpoints.

For closed curve, lines are adjusted to be tangent to the curve boundaries, that is, the line is snapped to the curve at only one intersection point. Figure 10 shows an example of the adjustment made on such junction.



(a) (b) (c) (d)

Figure 10: Intersection adjustment on full ellipse for line and curve junction (a) Initial sketch; (b) Fitted result; (c) Before adjustment; (d) After adjustment.

5.3 Parallelism Correction

After the endpoints clustering process, the edges of the object can be represented by single 2D geometry, in perfect straight-lines or ellipses that are joined to each other. To allow some tolerance for freehand sketching error, straight-lines are tested if they have similar gradients or orientations. A slight change in the orientation of lines can be done to ensure parallelism in the sketch, which is often difficult to be achieved by freehand sketching.

Each straight-line edge is tested against all the other edges to obtain their similarity with each other. Edges drawn with more number of strokes are determined to be more important and such, are used as the reference. All adjustment of an edge is achieved by rotating at the middle point of the edge. Figure 11 shows an example of the parallelism correction.

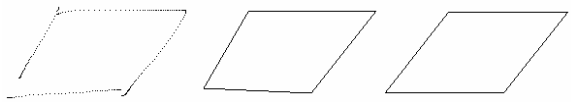


Figure 11: A rectangle before and after parallelism correction.

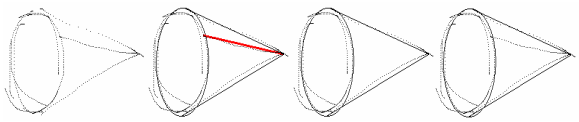
If the parallelism correction process changes line endpoints, the clustering process (as discussed in 5.1) will be carried out for the associated strokes to ensure the junction connectivity.

6. IN-CONTEXT INTERPRETATION OF STROKE

The calligraphic interface system developed here is meant as a preliminary system taking in 2D sketches, which are then interpreted so that they can later be used for 3D models reconstruction. As such, the system's interpretation is set to process sketches of 3D geometric objects drawn on 2D. We consider the isometric projection, although other projections can be used as well.

For a 2D sketch, a closed loop line or curve is used to represent a face of a 3D geometric object. A closed loop is normally interpreted as a surface, depending on the shape of the object. A complete 3D object should have no open edge in sketch. Any stroke with endpoint unconnected to a junction in the sketch will be considered as an error, and the stroke will be tagged as ambiguous. The system will highlight the strokes and prompt the user with a choice to either delete or keep them. If the user decides not to delete the strokes, the system will reinterpret the strokes based on the context so that they satisfied the geometry constraints.

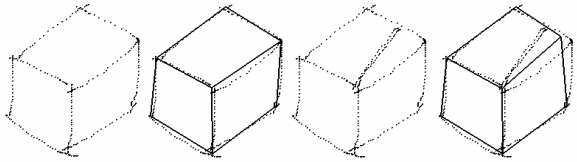
There are two sources for the ambiguous strokes. Firstly, the strokes can be caused by the misclassification of strokes at the beginning of the interpretation. For example, a short curve drawn with high sketching speed can be misinterpreted as a line by the classification method. To correct it, the system will reclassified it as curve, and continue with the grouping and fitting with other strokes in the sketch. Secondly, the stroke might be caused by poor sketching skill of the user, where overtracing strokes that meant to be grouped into an edge are sketched too far apart. The overtracing strokes failed the grouping tests and were grouped into separate edge. The both edges have the same endpoints that can be detected by our system, and grouped into one edge. Similar restriction applies to cylindrical object, where straight-lines only intersect curve at the boundaries of the curve surface (Figure 12).



(a) (b) (c) (d)

Figure 12: Lines detected as ambiguous with endpoints at an arc: (a) Input sketch; (b) Detected ambiguous stroke is in thick red color; (c) Result when user choose to delete the strokes; (d) Result when user choose to merge the strokes.

The in-context interpretation also allows the user to modify sketch with overtracing strokes (Figure 13). The system will try to connect 'open' strokes in the sketch to its nearest junctions, resulting regrouping and refitting of the strokes. The fitted result is therefore moved by the new strokes and modified the initial sketch.



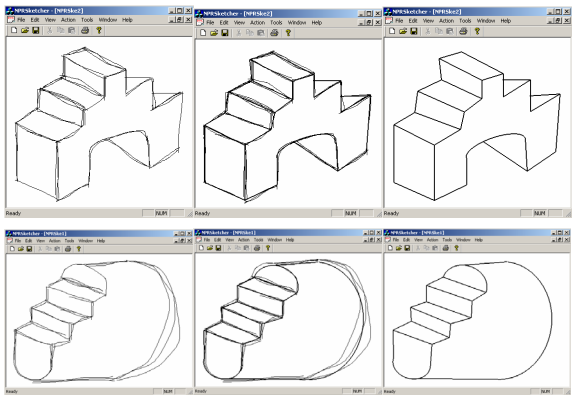
(a) (b) (c) (d)

Figure 13: Modification by overtracing; (a-b) Initial sketch with fitted result; (c-d) Sketch with 'open' strokes and new fitted result.

7. IMPLEMENTATION AND EXAMPLES

The system is implemented using Visual C++ under the Windows 2000 operating system. The input device is a traditional digitizing tablet that senses the drawing on its screen.

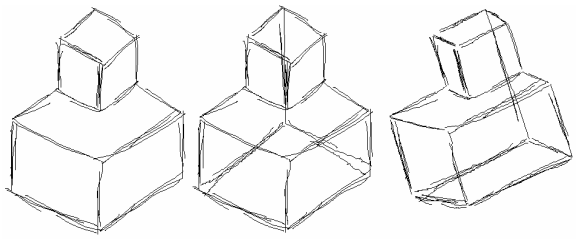
Figure 14 shows some examples of the sketch before and after tidy-up with our system. The interface allows the user to sketch freely as though using pen and paper. The system will interpret and tidy-up the sketch into a single line drawing.



(a) (b) (c)

Figure 14: (a) Input sketch; (b) Tidied-up sketch; (c) Single-line output.

Our system keeps the initial sketch information and associates them with the corresponding edges in tidy-up drawing. The information can be used to render additional new strokes in the sketch that are of similar appearance to the original sketch even though the new strokes are not actually drawn by the user. This suggests that the result from our system can be used as input for non-photorealistic rendering (NPR) system that renders 3D objects with sketchy appearance that is similar to the original sketch by the user. Figure 15 shows the result of such an implementation.



(a) (b) (c)

Figure 15: (a) Initial sketch; (b) 3D model in NPR; (c) 3D model after rotation.

8. CONCLUSION

This paper presents an interactive calligraphic interface for conceptual design, which supports multistroke sketching. It is designed to handle freehand sketches with overtracing and imperfections. User can sketch over the existing drawing to enhance, complete or correct an edge. The system provides in-context interpretation for sketch that depict 3D geometric object. The interpretation provides verifications and corrections for errors made during sketching session, thus achieving meaningful tidy-up result.

We proposed an alternative interpretation of overtracing strokes from freehand sketch input from the existing systems. We consider all strokes as part of the sketch and grouped them into edges. We introduce a new method to group multiple curves using the minimal enclosing bounding box. The method is fast and hence suitable for online sketching.

The work presented here is only the first part of our final sketched-based 3D modeling and rendering system. The freehand sketch input is processed and tidied-up for the 3D reconstruction. The reconstructed 3D model will have a photorealistic appearance by default. The future work is to reconstruct and display the 3D model with the appearance similar to the original sketch in a non-photorealistic rendering form.

9. REFERENCES

[BOW83a] Bowyer, A., and Woodwark, J., A Programmer's Geometry, Butterworths, 1983.

[FIO02a] Fiore, F.D., and Reeth, F.V., A Multi-Level Sketching Tool for Pencil-and-Paper Animation, AAAI 2002 Spring Symposium., Tech. Rep. SS-02-08, 2002.

[GRO00a] Gross, M.D., and Do, E.Y., Drawing on the Back of an Envelope: a framework for interacting with application programs by freehand drawing, Computers & Graphics, vol.24, pp.835-849, 2000.

[IGA98a] Igarashi, T., Kawachiya, S., Tanaka, H., and Matsuoka, S., Pegasus: a drawing system for rapid geometric design, ACM Press, pp. 24-25, 1998.

[IGA97a] Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H., Interactive beautification: a technique for rapid geometric design, ACM Press, pp.105-114, 1997.

[JEN92a] Jenkins, D.L., and Martin, R.R., Applying constraints to enforce users' intentions in free-hand 2-D sketches, Intell.Syst.Eng., vol.1, pp.31-49, 1992.

[KAR05a] Kara, L.B., and Stahovich, T.F., An image-based, trainable symbol recognizer for hand-drawn sketches, Computers & Graphics, vol.29, pp.501-507, 2005.

[MIT02a] Mitani, J., Suzuki, H., and Kimura, F., 3D sketch: sketch-based model reconstruction and rendering, Kluwer Academic Publishers, pp.85-98, 2002.

[ONE83a] O'Neil, P.V., Advanced engineering mathematics, Wadsworth Publishing, pp.1091-1093, 1983.

[PAV85a] Pavlidis, T., and Wyk, C.J.V., An Automatic Beautifier for Drawings and Illustrations, SIGGRAPH., vol. 19, pp.225-234, 1985.

[QIN01a] Qin, S.F., Wright, D.K., and Jordanov, I.N., On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations, Pattern Recognit, vol.34, pp.1885-1893, 2001.

[QIN00a] Qin, S.F., Investigation of Sketch Interpretation Techniques Into 2D and 3D Conceptual Design Geometry, University of Wales Institute, Cardiff, PhD Thesis, 2000.

[QIN00b] Qin, S.F., Wright, D.K., and Jordanov, I.N., From on-line sketching to 2D and 3D geometry: A system based on fuzzy knowledge, CAD Computer Aided Design, vol.32, pp.851-866, 2000.

[QIN99a] Qin, S.F., Jordanov, I.N., and Wright, D.K., Freehand drawing system using a fuzzy logic concept, CAD Computer Aided Design, vol.31, pp.359-360, 1999.

[SEZ01a] Sezgin, T.M., Stahovich, T., and Davis, R., Sketch based interfaces: early processing for sketch understanding, PUI '01: Proc. of the 2001 workshop on Percetive user interfaces, pp.1-8, 2001.

[SHE04a] Shesh, A., and Chen, B., SMARTPAPER: An Interactive and User Friendly Sketching System, Comput.Graphics Forum, vol.23, pp.301-301, 2004.

[SHP97a] Shpitalni, M., and Lipson, H., Classification of sketch strokes and corner detection using conic sections and adaptive clustering, J Mech Des, Trans ASME, vol.119, pp.131-135, 1997.

A simple construction method for sequentially tidying up 2D online freehand sketches

Shengfeng Qin
School of Engineering and Design
Brunel University, Uxbridge
Middlesex, UB8 3PH, UK
Sheng.feng.qin@brunel.ac.uk

David K Wright
School of Engineering and Design
Brunel University, Uxbridge
Middlesex, UB8 3PH, UK
David.wright@brunel.ac.uk

Ivan Jordanov
School of Computing
University of Portsmouth
Portsmouth PO1 3HE, UK
Ivan.Jordanov@port.ac.uk

ABSTRACT

This paper presents a novel constructive approach to sequentially tidying up 2D online freehand sketches for further 3D interpretation in a conceptual design system. Upon receiving a sketch stroke, the system first identifies it as a 2D primitive and then automatically infers its 2D geometric constraints related to previous 2D geometry (if any). Based on recognized 2D constraints, the identified geometry will be modified accordingly to meet its constraints. The modification is realized in one or two sequent geometric constructions in consistence with its degrees of freedom. This method can produce 2D configurations without iterative procedures to solve constraint equations. It is simple and easy to use for a real-time application. Several examples are tested and discussed.

Keywords

Sketch-based design, Geometric constraints solver, Computer-human interface.

1. INTRODUCTION

In a conceptual design stage, product design and development often takes the form of the artist's sketches. In order to reduce product lead-time, transition directly from the stylist's sketches to a computer model is desirable [Zel96]. To meet this need, research has been carried out to develop a sketch-based user interface, recognize 2D primitives through a 2D sketch segmentation, classification and identification process and infer 3D objects [Qin00, Qin01]. Recognized 2D primitives include straight lines, circles, circular arcs, ellipses and elliptical arcs, or B-spline curves. These 2D entities are fitted with least square algorithms, but in general, they are not connected properly to reflect the user's intention. Modification in 2D is therefore required in order to give them proper position, direction and connections among them. Identification of various 2D constraints such as connectivity, parallelism and perpendicularity, is prerequisite for the 2D modification and further 3D interpretation.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

This paper presents a novel and simple constructive approach to beautifying 2D geometry based on freehand sketches. It includes three parts: (1) inferring 2D geometric constraints from rough sketches; (2) finding a solution to satisfy the constraints wherever possible; and (3) finally modifying drawing to a desired 2D geometry. The approach is based on constructive principles and degrees of freedom analysis.

This paper is organized as follows. Section 2 reviews related works and Section 3 shows constraints classification and capturing. Sections 3 and 4 describe the analysis of degrees of freedom for objects and constraints. Section 5 discusses the constructive rules. Finally, examples and conclusion are given in Section 6 and 7.

2. RELATED WORKS

Shpitalni and Lipson [Shp97] presented an approach for classifying pen strokes in an on-line sketching system and an adaptive method for clustering disconnected end-points. The following steps are applied by the clustering algorithm: (1) creating raw vertices at all endpoints of entities in the drawings, (2) determining the radius of the tolerance circle around each raw vertex, (3) identifying and grouping pairs of raw vertices when each member of the pair falls within the other members tolerance circle, (4) iteratively grouping chains of pairs into clusters and

finally (5) placing a vertex node at the centroid of each cluster and adjusting lines and arcs accordingly. This method is only concerned about coincidence constraint among endpoints. It needs to wait for a completion of sketch input and then starts to tidy up. The inaccuracy of interpretation may increase. Furthermore, taking the centroid of each cluster and adjusting geometry accordingly may change some relations such as parallelism.

An automatic beautifier for drawings and illustrations was studied by Pavlidis and Van Wyk [Pav85]. A method was developed for inferring constraints that are desirable for a given (rough) drawing and then modifying the drawing to satisfy the constraints wherever possible. Drawings here were polygon-oriented. The relations (constraints) examined are: approximate equality of slop or length of sides (line segments), collinearity of sides, and vertical and horizontal alignment of points. The system restricted the number of constraints to avoid an explosion in processing time. The solution of the constraints is not always guaranteed.

A similar system Easel [Jen92] was developed by Jenkins and Martin. It behaves in as nearly an automatic manner to infer constraints and then tidy up the drawing. Geometric entities include straight lines, circular arcs and composite Bezier curves. Relations consist of unary relations such as close to a point and pairwise relations. The constraints are satisfied with multiple enforcements based on scenario analysis. Easel's performance is simply not good enough for practical sketches consisting of perhaps hundred of elements because of a time delay.

In general, geometric constraints can be topological (structural) ones, such as incidence, tangency, parallelism, perpendicularity, etc., or metric (dimensional) ones, such as distance or angle. When solving geometric constraints, a solver must produce an instance of given topology (structure) that exactly satisfies given constraints. The main geometric constraint solvers can be divided into two categories: and constructive solvers. The equational solvers translate geometric constraints into a system of algebraic equations, which is then solved using different iterative techniques. These solvers are based on numerical methods [Jen92] and symbolic methods [Gao98]. The shortcomings of numerical methods include slow runtimes, numerical instabilities and difficulties in handling redundant constraints. The disadvantage of symbolic solvers is that they are still too slow for real-time computation [Li02].

The constructive solvers [Bou95] make use of the fact that most configurations in an engineering drawing are solvable by ruler, compass or protractor. A planning phase is carried out to transform a constraint problem into a step-by-step constructive form that is easy to compute, and then the constraint system can be solved efficiently. Generally speaking, the above solvers rely heavily on the user interaction to produce constraints either by stating relations, or by adding dimensions. These systems also focus on *rc-configuration* (ruler and compass) problems in which primitives such as ellipses and elliptic arcs are excluded.

Our system can automatically infer constraints during sketching with its inference engine, and then modify drawing in one pass to give one of the possible solutions to the constraints. Therefore, it is simple and easy to use for on-line applications.

3. CONSTRAINT INFERENCE ENGINE

In our system, once a stroke has been sketched out, it will be segmented into a series of head-to-end connected sub-strokes if necessary. Each sub-stroke will then be classified and fitted with one of 2D primitives: straight lines, circles, ellipses, circular arcs, elliptic arcs and B-spline curves [Qin01]. After the closest fitting has been found, the system constraint inference engine will infer certain geometric constraints. They can be classified into three categories: *unary*, *pairwise*, and *connection* constraints [Gao98]. The engine will first search for unary constraints and then establish pairwise and connection constraints by checking its relations to previous strokes (or sub-strokes) backwards sequentially. Once the current stroke becomes over-constrained, the inference process will be stopped and then the constraints solver will generate construction steps to solve the identified constraints.

Unary Constraints

The unary constraints are properties of a single primitive. They are directional constraints. The unary constraints apply to straight lines, ellipses, and elliptical arcs. For a straight line, the engine examines its directional angle to see whether it is roughly horizontal, or vertical, or parallel to isometric projection axes (Fig.1). If so, the straight line will be assigned corresponding unary constraint code: HOR, VER (or ISO-Y), ISO-X or ISO-Z. Similarly, for an ellipse, the system checks its major axis. For an elliptical arc, the system still checks its direction of the major axis, as for an ellipse. The rule for determining a unary constraint is that the directional angle (α) of a primitive is within a range of (β-δ) and (β+δ), where β is a constraint angle in degree (0 for HOR, 90 for VER, 30 for ISO-X and –30 for ISO-Z) and δ is an adaptive tolerance angle for the primitive. That is, $(\beta-\delta) < \alpha < (\beta+\delta)$. The parameter δ varies

with drawing speed and sizes of primitives such as lengths of lines or major axis of an ellipse. The bigger the sizes are the bigger δ. The higher the speed is the bigger δ.

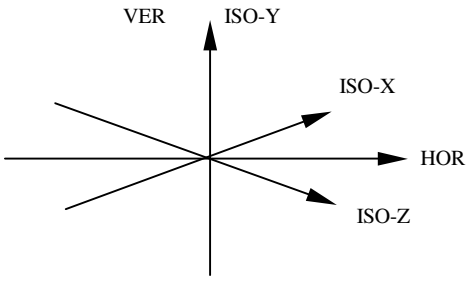


Figure 1. Directional Constraints

Pairwise Constraints

The pairwise constraints are geometric relations shared by two primitives [Gao98]. Currently, the system supports parallelism and perpendicularity between pairs of lines, ellipses, or elliptical arcs. Each line, or ellipse, can have one of the pairwise relations: either parallelism, or perpendicularity (it may have both, but the system only takes one of them). In essence, these two relations are directional constraints as well. The system searches these relations backwards for the current primitive (the latest input) by comparing directions of the current primitive and one previous primitive. If they are quite close, a parallelism relation will be found. If the difference between their directions is close to 90 degree, a perpendicularity relation will be determined. Once a relation is found, the system will stop a further search, otherwise, the search will continue until the first primitive is reached. This will reduce the number of constraints and avoid over-constrained cases. For example, a line A is parallel to lines B and C. If a parallelism relation between the lines A and B is found, then the relation between the lines A and C will not be further checked because the lines B and C should be parallel to each other. Similarly, if a line D is parallel to a line E and perpendicular to a line F, the system will only take the first found constraint because that the two constraints should be consistent, one is enough for constraining a direction.

Connection Constraints

The connection constraints are classified into three categories, namely *type*-1, 2, and 3 according to typical application scenarios.

Type-1 constraints

From the current primitive to a previous one, the inference engine searches for connectivity relations. For a *type*-1 constraint, two primitives intend to join together at their end points. An example is shown in Fig. 2. The engine will first search for a pair of end points between two primitives and then check whether their end tolerance circles have intersections. If so, the two primitives will be connected at related end points. The radius of an end tolerance circle for a line is adaptive to its length and drawing speed. The longer the length is the bigger the radius. The higher the speed is, the greater the radius. Similarly, for an arc, the tolerance varies with its arc length and drawing speed. If an end point is constrained with a *type*-1 relation, the relation code 1 is assigned to it (default is 0, meaning free end) and the corresponding constraint information will be recorded. Here, an adaptive tolerance is applied, since a simple fixed value may be too large or too small, resulting in either eliminating fine details in connections, or leaving adjacent ends unlinked. Indeed, different tolerances are needed for different parts of sketches, and certainly for different users. Using the adaptive tolerance can roughly satisfy this requirement.

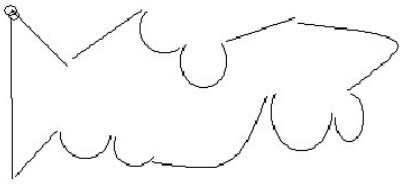


Figure 2. Type-1 constraints

Type-2 Constraints

A *type-2* constraint is a touching relation, in which an end point of a primitive falls on the path of another (Fig. 3). This constraint is only applied to lines and arcs. That is, a primitive joins another with its one end touching on another in the middle. The constraint code for this relation is 2. To detect a *type*-2 relation, the following procedure is conducted:

Step 1: Compute an adaptive tolerance value for the current primitive;

Step 2: Check if the corresponding tolerance circles at ends are intersected with a candidate primitive; if not, search for another primitives;

Step 3: If so, a type-2 constraint is found and a constraint code 2 will be assigned to the corresponding end and related constraint information will be stored.

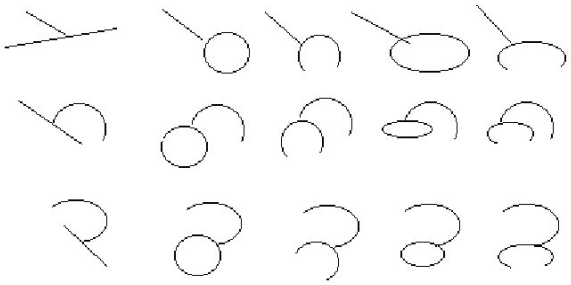


Figure3. Type-2 constraints

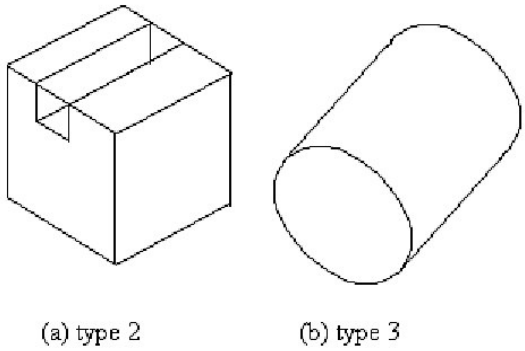


Figure5. Examples of different constraints

Type-3 constraints

The third type constraint (relation code 3) deals with a tangent connection, as shown in Fig.4, in which one end of a primitive is tangent to another primitive. Such a constraint is only concerned with lines and arcs as well. This connection can be regarded as a special case of a type-2 constraint. The current primitive not only joins the other with one end but also is tangent to it.

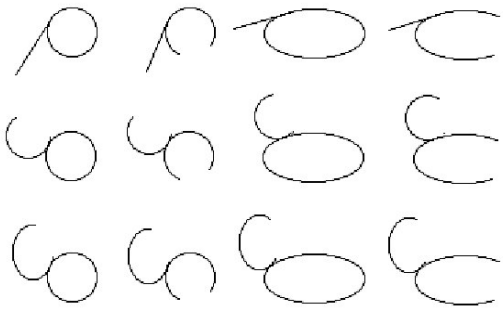


Figure 4 *Type-3* constraints

To determine a *type-3* constraint, two steps are applied. First step is to check if the current primitive has a type-2 constraint. If so, the next is to further determine whether the connection meets a tangent condition. Taking a pair of a line and a circular arc as an example, we can recognize that the connection is tangent, if the distance between the line and the centre of the arc is close to the radius of the arc. Once a *type*-3 constraint is found, the former relation code 2 for a type-2 constraint will be updated to a relation code 3. The corresponding constraint information will be recorded. Fig.5 gives some examples of different connections. When sketching a slot feature from a box, users will meet first and second type constraints (Fig.5 (a)). When silhouette lines are drawn to express a cylindrical object (Fig.5 (b)), the third type constraint will occur.

4. DEGREES OF FREEDOM ANALYSIS

Once a variety of constraints (relations) are obtained, the next is to modify individual primitives to satisfy all constraints, or to find a satisfactory solution. The system first analyses the degrees of freedom (DF) of a primitive and then determines construction rules for the primitive under certain constraints, using the degrees of freedom analysis.

Definitions

Informally, the number of degrees of freedom of a primitive object (object degrees of freedom, ODF) is the number of independent parameters required to allow the primitive to vary in location and shape. For example, in 2D space, a rigid body has two translational and one-rotational degrees of freedom to change its location. But, for ODF, extra degrees of freedom are allowed to vary its shape as well. For instance, a 2D arc may have extra 3 object degrees of freedom in terms of starting angle, subtended angle and radius to change its shape. Note that only the subtended angle and radius cannot define the starting point on the arc.

The number of constrained degrees of freedom (CDF) from a constraint is the number of degrees of freedom eliminated by the constraint. For instance, in 2D space, a position constraint limits two translational degrees of freedom of a primitive. Under given constraints, a geometric constraint solver may configure a primitive in limited ways. This is regarded as configuration degrees of freedom (CF), which is the difference between the number of ODF and the sum of its corresponding CDF. The relationship among ODF, CDF and CF can be addressed as

$$CF = ODF - \sum_{i=1}^{n} CDF_i ,$$

where n is the number of constraints. We consider a primitive as well defined (or well-constrained), if and only if CF is equal to zero. It is under-constrained, when CF> 0, and over-constrained while CF < 0.

Object Degrees of Freedom

In our system, there are no explicit dimensional constraints. Thus, each primitive may vary in location or shape. This means that primitive geometry in the system is not a rigid-object. Different primitives have different object degrees of freedom, in accordance with different construction limitations. The degrees of freedom (DF) for each primitive are shown in Table 1. For example, a circle has no rotational DF because it is a perfect symmetry; it also has no dimensional DF, which means that its radius is fixed during construction processing. This assumption will make the construction task simple. Similarly, this dimensional restriction is applied on ellipses. However, for an ellipse, a rotational degree is given to allow a rotation of its major axis about its centre for meeting a directional constraint. Although an ellipse can be constructed by its four correspond circular arcs using *rc*-configurations, its rotational degree of freedom is unique comparing to a circle. For a circular or an elliptical arc, a dimensional DF is given to allow the system to change its extended angles, but not for changing its radius (or radii). Here the system simply treats a B-spline curve like a straight line. Note that lines, arcs and B-sline curves have the same structure of object degrees of freedom. If they are under the same constraints, their constructions rules will be similar.

Primitives	Translational DOF	Rotational DOF	Dimensional DOF	Total ODF
Line	2	1	1	4
Arc	2	1	1	4
Elliptical arc	2	1	1	4
Circle	2	0	0	2
Ellipse	2	1	0	3
B-spline	2	1	1	4

Table 1. Object degrees of freedom

Constrained Degrees of Freedom

Various constraints restrict different degrees of freedom. The constrained degree of freedom (CDF) for each type of constraints is given in Table 2. A pairwise or unary constraint will restrict a rotational degree of freedom. For a *type*-1 relation, it is an incidence constraint, which restricts two translational degrees of freedom. For a *type*-2 relation, it requires that one end point of a primitive to be extended onto a constrained primitive. So, this type constraint eliminates a dimensional degree. A *type*-3 constraint will remove a dimensional degree of freedom as a type-2 one, and further restrict a rotational degree of freedom by requiring a tangency relation.

Constraints	CDF
Pairwise	1
Unary	1
Type 1	2
Type 2	1
Type 3	2

Table 2 Constrained degrees of freedom

5. CONSTRUCTION RULES

To configure sketched 2D primitives with identified constraints, the system calculates configuration degrees of freedom and then produces construction rules (or steps) according to several general construction strategies.

General Construction Strategies

When solving constraints, the following general construction strategies are applied to all types of primitives to generate construction steps:

(1) If a primitive is free from any constraints, default constraints for fixing its position will be applied. In this case, its CF is zero.

(2) If a primitive of lines, ellipses or elliptic arcs has a unary constraint and it is well-constrained or under-constrained, *Minimal movement policy* will be applied on it. That is, if the current element is required to change its direction, the system should try its best to keep movements minimal, since original position and size of the geometry represents users' initial intent. This policy attempts to capture users' intent more accurately. Fig.6 gives an example of this minimal movement strategy. In Fig.6 (a), a straight line (dashed line) needs to be modified to a vertical line. In accordance with the current policy, the solver rotates it about its mid-point, to a vertical line (solid line). The system does not take the second solution (Fig.6 (b)), which rotates the line about its one end to form a vertical line, because the resulting line will be far from the original one.

Figure 6. Minimal movement

(3) *One-side policy:* when dealing with the current primitive, the constraint solver ignores all constraints between the current primitive and those generated after it. This means that only constraints between the current primitive and previous ones (on one-side of it) will be solved. This strategy reduces the number of constraints to be treated, and focuses on a local configuration problem. This policy respects the fact that when sketching a current stroke, the user mainly takes previous drawing as references to form new constraints, although some intentions might be born at this moment. If the user stops drawing after the current stroke, the system should still give a possible solution.

(4) Background propagation: if a current primitive has some constraints with previous ones, the solver first tries to modify it to satisfy the constraints, and to keep previous ones unchanged, although the constraints could be met by changing the previous, either. Otherwise, once a new stroke inputs, some new constraints may be added in a constraint chain from the current primitive to the first one and all previous geometry will be changed wavelike. This will not only lead to heavy computation and instability, but will also harm the minimal movement policy. Actually, the backward propagation strategy can be introduced from the minimal movement policy.

(5) Clustering policy: if any two primitives meet at their end points by a *type*-1 constraint, the common position will be figured out and fixed for ever. If none of them has a directional constraint, the common position will be a mid-point of related two ends. If any of them has a directional constraint, their geometric intersection point will be the common position. Once a common position is found, it will become fixed.

Generation of Construction Steps

Before considering how to configure a new primitive, analysis of its constraints and degrees of freedom is performed. Then the decision on how to construct the geometry in sequential steps is made accordingly. In general, if geometry is under-constrained, a set of default constraints will be applied to make it well-constrained. Afterwards, any default constraints can be further modified such as free ends. The most common default constraint is a joint restricting two translational degrees of freedom. When the geometry becomes well-constrained, it can be constructed against typical application scenarios. If it is over-constrained, typically, extra constraints such as directional ones will be removed to make it well connected. In our system, connection constraints have a higher priority than directional ones because they contain more important topological information for 3D interpretation.

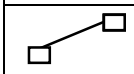
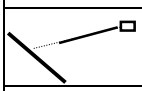
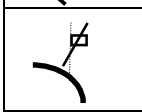
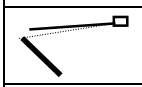
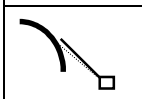
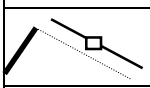
Scenario	No.	Constraints U-unary, P-pairwise	ODF	CDF	CF	Add Default constraints □ J-Joint D-Dimension
	1	0	4	0	4	2 J
	2	1 U or 1 P	4	1	3	1 J 1 D (explicit)
	3	1 *type*-2	4	1	3	1 J 1 D (implicit)
	4	1 *type*-2 and 1 U or 1 P	4	2	2	1 J 1 D (implicit)
	5	1 *type*-1	4	2	2	1 J
	6	1 *type*-3	4	2	2	1 J
	7	1 *type*-1 and 1 U or 1 P	4	3	1	1 D (explicit)
	8	1 *type*-3 and 1 U o1 P	4	3	1	1 D (explicit)

Table 3. General construction analysis-under constrained cases

Tables 3 and 4 illustrate a case study for a line configuration. In general, there are 13 combinations of different constraints. Most of them (nine cases) are under-constrained (see Table 3). Only three of them are well constrained in Table 4 (No. 9-11) and the last two cases are over-constrained. This means that for most of the cases, a possible solution can be found easily under the current solving strategies. The main concern is how to add default constraint(s) and solve constraint equations. After choosing the default constraints the construction steps will become well defined. If the constraints include a directional one (unary or parallelism or perpendicularity), in general, two construction steps are needed. They can be performed separately by firstly modifying the current primitive to meet the directional constraint and then simply focusing on the predefined a one-step construction. For example, in the case No.4 (Table 3), a line is constrained with a unary relation (VER)

and a type-2 relation. We can solve the problem in two steps. First rotating the line around its mid-point to meet the unary relation. This operation is the same as in the case No.2 with a default explicit dimensional constraint. After that, the problem will be similar to the case No.3. The second step is to extend the line by finding its intersection point to meet the type-2 constraint, which can be regarded as an implicit dimensional constraint. In these three cases, program routines for changing direction and obtaining intersections are separate. They are reusable and combinable. This can not only save developing time, but also reduce the number of constructive steps. Case No. 1 simply takes two default ends and Case No. 5 takes one default end and a constrained end with clustering. Case No.6 can be solved by finding a tangent line from the default end. For cases No. 7 and 8, after a rotation, the next is to move the line to an incident point or tangent point. For three well-constrained cases (Table 4), they need only one step to solve their constraint equations, which depends on the types of involved primitives. The last two cases (Table 4) are over constrained, they can be first modified into well-constrained cases and then solved in a similar way to Case No. 9.

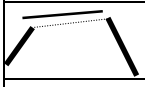
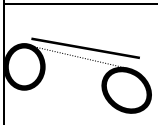
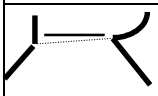
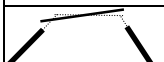
Scenario	No	Constraints U-unary, P-pairwise	ODF	CDF	CF	Remove Default constraints
	9	2 *type*-1	4	4	0	0
	10	1 *type*-1 and 1 *type*-3	4	4	0	0
	11	2 *type*-3	4	4	4	0
	12	2 *type*-1 and 1 U or 1 P	4	5	-1	1 U or 1 P
	13	2 *type*-1 and 1 U or 1 P.	4	5	-1	Modify free ends or remove 1U or 1P

Table 4 General construction analysis -well and over constrained cases

B-spline curves have been restricted to have only *type-1* constraints. They can be just regarded as special cases of lines, as in cases No. 1, No. 5 and No. 9. The solver simply assigns incident points to their end points. A circle can only move in 2D with a constant radius. So, its construction is always to find a displacement of its centre point. An ellipse direction can be changed under a unary or a pairwise constraint, and also its centre points can be shifted in a similar way to a circle.

Circular and elliptical arcs are open curve sections with two ends, and have 4 object degrees of freedom as lines. They also have the same types of constraints to be applied as lines. Topologically speaking, they are within the same class of line objects for tidying up. Therefore, construction rules for arcs are similar to those for lines. Each line case has a corresponding case for arcs. Taking the case No. 3 as an example, the construction method for a line is to find its intersection point between two lines. For an arc (circular or elliptical), the construction method is still to find intersection point, but between an arc and a line. The difference is the use of different equations to obtain an intersection point. But, the construction method is the same.

5. EXAMPLES AND DISCUSSION

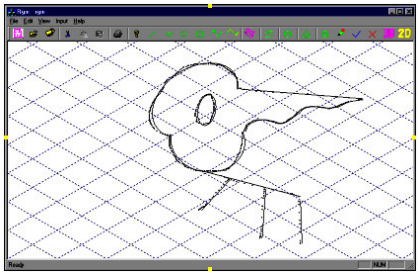


Figure 7. Input of sketches

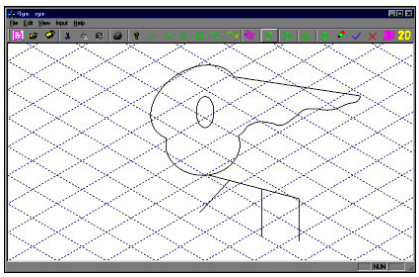


Figure 8. Tidying up

With our system, 2D online sketches can be rapidly transferred into 2D primitives and further can be beautified with right connections. The tidying up processing is based on our construction rules and degrees of freedom analysis. This method lets users to work on their design ideas with a real-time system in a more natural way. Fig. 7 illustrates original input of sketches, which consists of several lines, arcs and a B-spline curve. Constraints involved in this case include *type*-1 and *type*-2 connections, e.g., a line touching an arc, and unary relations, e.g. vertical

lines. These constraints are detected successfully by the inference engine, and then are solved properly by the constraint solver. Fig. 8 gives the result of this 2D configuration. It can be seen that the ellipse in the middle and the two vertical lines are changed under unary constraints. Three lines are modified to touch on other primitives under *type*-2 constraints. All *type* –1 constraints are solved correctly.

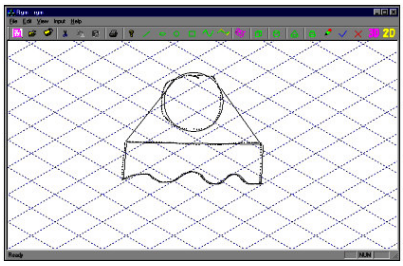


Figure 9. Sketched input

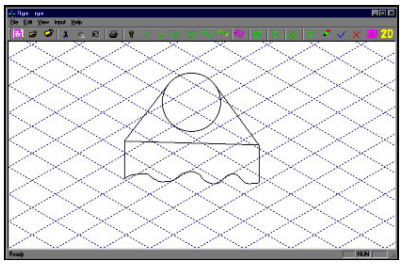


Figure 10. Result of beautification

Figure 9 shows an example of sketches with *type*-3 constraints. Two lines tangent to a circle are inputted. The result of the tidying up is given in Figure 10. The last stroke for the horizontal line is an over-constrained case. Its unary constraint (HOR) is removed off because its two ends become fixed points already. It can be seen that the system can capture *type*-3 constraints (tangency), and the solver works properly.

7. CONCLUSION

The constraint solver based on the construction rules and degrees of freedom analysis can quickly and properly give one of the possible solutions. It determines primitives one by one, and does not involve solving a system of simultaneous non-linear equations. The inference engine and the constraint solver can deal with elliptical primitives and free-form curves.

The System works directly on sketches. No dimensional schema is required and users are not asked to add dimensions to the sketches, as in most commercial parametric CAD systems. The constraint solver treats 2D primitives as semi-rigid-objects with a dimensional degree of freedom. For example, a line in 2D space has 4 object degrees of freedom instead of 3 for a rigid-body. In this way, the solver treats the dimension information either as default constraint (changeable constraint), depending on the object's configuration degrees of freedom. In contrast, most geometric constraint solvers [Gao98,Bou95], regarded dimensional constraints as rigid constraints.

The system is performed fast enough for a real-time sketch-based application. This is important for conceptual design, especially for distributed design systems [Qin03]. The solver will require more pre-coded construction rules, if the number of constraint types is increased. This drawback will be balanced with its speedy performance.

8. REFERENCES

[Zel96] Zeleznik, RC, Herndon, KP, and Hugnes, JF, SKETCH: an interface for sketching 2D scenes, *SIGGRAPH,* pp.163-170, 1996.

[Qin00] Qin, SF, Wright, DK and Jordanov, IN, From on-line sketching to 2D and 3D geometry: a system based on fuzzy knowledge, *Computer-Aided Design* 32, pp.851-866, 2000.

[Qin01] Qin, SF, Wright, DK and Jordanov, IN, On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations, *Pattern Recognition* 34, pp.1885-1893, 2001.

[Shp97] Shpitalni M and Lipson H, Classification of sketch strokes and corner detection using conic sections and adaptive clustering, Journal of Mechanical Design 119, pp.131-135, 1997.

[Pav85] Pavlidis T, and Van Wyk CJ, An automatic beautifier for drawings and illustration, ACM Computer Graphics 19 (3), pp. 225-234, 1985.

[Jen92] Jenkins DL Martin RR, Applying Constraints to enforce users' intentions in free-hand 2D sketches. Intelligent Systems Engineering, Vol. 1, 31-49, 1992.

[Gao98] Gao XS, Chou SC, Soving geometric constraint systems II. A symbolic approach and decision of rc-constructibility, Computer-Aided Design 30(2), pp.115-22, 1998.

[Li02] Li YT, Hu SM, and Sun JG, A constructive approach to solving 3-D geometric constraint systems using dependence analysis, Computer-Aided Design 34, pp. 97-108, 2002.

[Bou95] Bouma W Fudos I, Hoffmann C, Cai J, Paige R, Geometric constraint solver. Computer – Aided design 27(6), pp. 487-501,1995.

[Qin03] Qin SF, Harrison R, West AA , Jordanov and Wright DK, A framework of web-based conceptual design, Computers In Industry 50, pp.153-164, 2000.

A New Approach to Urban Modelling Based on LIDAR

Rebecca (Oi Chi) Tse
rtse@glam.ac.uk

Christopher Gold
christophergold@voronoi.com

Dave Kidner
dbkidner@glam.ac.uk

School of Computing, University of Glamorgan, Pontypridd CF37 1DL, Wales, UK

ABSTRACT

Estimating building forms from LIDAR data has usually been done by attempting to fit standardized building types to the residual data points after an estimated bare earth terrain surface has been removed. We propose an approach based on segmenting the raw data into high and low regions, and then modelling the walls and roofs by extruding the triangulated terrain surface (TIN) using CAD-type Euler operators. The segmentation may be done by the addition of building boundary data to the TIN so as to force triangle edges to match the boundaries, and then using Euler operators to extrude the building, producing vertical walls rather than the more usual sloping walls formed from TIN models alone. If boundary data is not available then an automated segmentation method based on adaptive Voronoi cells may be used, so that each cell contains either high or low LIDAR data, but not both. This prismatic model, with flat-topped cells, approximates the building forms without hypothesizing specific building types. Once the segmentation is achieved, and the walls constructed, we attempt to model the roofs by calculating the eigenvalues and eigenvectors of the vector normals of the TIN sections within the building boundaries. The smallest eigenvalue gives a predicted roof orientation, and the resulting roof profile is then modelled.

Keywords
Urban modelling, LIDAR, Euler operators

1 INTRODUCTION AND PREVIOUS WORK

LIDAR (Light Detection and Ranging) data is widely used to construct 3D terrain models to provide realistic impressions of the urban environment and models of the buildings. The latest airborne laser scanning technology allows the capture of very dense 3D point clouds from the terrain and surface features. The most common method is to remove all the buildings, trees and terrain objects and generate a bare-earth model [Vos03]. Building boundaries are obtained and estimated building forms are reconstructed by CAD software and pasted on top of the bare-earth model. This approach is sufficient for visualization, but inadequate if the resulting model is to be used for analysis: flood, simulation, trafficability, etc. We therefore wish to integrate our buildings with the terrain surface. A second concern is the source of building outlines: if they are not available from map data they must be extracted from the LIDAR data itself. This is usually imprecise. Suveg and Vosselman [SV04] give an extensive survey of building modelling, but based on photogrammetric data.

Conference proceedings ISBN 80-86943-03-8
WSCG'2005, January 30–February 3, 2006
Plzen, Czech Republic.

Thus, in section 2, we use a traditional method to filter all the terrain objects, generate a bare-earth model and put Ordnance Survey Landline data as the building outlines on the terrain surface. Where map data is not available, building outlines must be extracted directly. Image analysis techniques may be used to identify linear features, but these are difficult to connect to form complete buildings. We describe a new alternative region-based approach where Voronoi cells are adjusted to contain either "building" or "non-building" LIDAR data points.

Section 3 describes how topologically integrated buildings may be extruded from the landscape, based Euler operators. This permits interactive editing of the 3D model [TG01].

Having constructed the basic flat-roofed building, Section 4 describes a method of reconstructing the roof shape from the interior LIDAR data. These points are first triangulated, and the triangle orientations pre-

served. Unlike Hofmann, Maas and Streilein [HMS03], who used slope, azimuth and a distance measure, we project the vector normals of the triangles onto the unit hemisphere, simplifying our clustering algorithms and allowing estimation of the roof axis, if present.

2 Methodologies for Building Reconstruction Using LIDAR data

We mentioned two segmentation methods which are:

- Landline Tracing
- Voronoi City Modelling

2.1 Landline Tracing

We use Ordnance Survey (OS) Landline, original and filtered LIDAR data points to create the urban model. In this paper we use the University of Glamorgan campus as an example. The original and filtered data were captured by Infoterra in March 2003. A filtering algorithm is used to remove all terrain objects, for example, buildings, trees and cars. Morphologic filtering is one of the most common algorithms used to create a bare-earth model, for example slope based filtering [Vos03] and modified slope based filtering [Rog02].

Several steps are used to reconstruct buildings using Landline Tracing. They are:

- Create a TIN model with the filtered LIDAR data
- Add Landline data to the terrain surface using our line tracing algorithm
- Use our point-location algorithm to find out the LIDAR data points for each building and calculate the average heights (z-value) of each building
- Extrude the building using CAD-type Euler Operators. The details will be shown in the next section

There are several steps in the line-tracing algorithm:

1. Insert two points on the terrain surface (Points A and B in figures 1)
2. Check if any previous triangle edge connects these two points
   - Stop if this is true, or
   - Insert a point half way between these two points if no edge connects them.

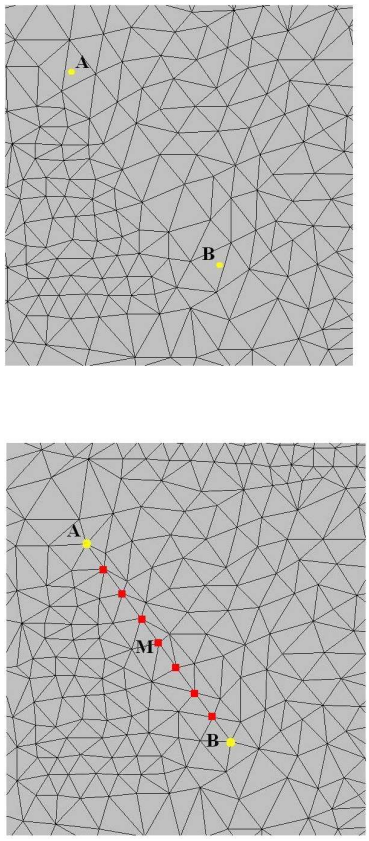


Figure 1: Connects points A and B by adding several (square shaped) points in between

3. Repeat step 2 for each half of the line recursively until points A and B are connected by triangle edges.

We add points on the surface and estimate the height of the points by using the surface interpolation method of [DG02].

We use the point-location algorithm to find the LIDAR data points which are inside the building boundary and to find the average height of each building.

The steps of the point-location algorithm are:

1. Delete all points inside the building boundary
2. Order the vertices of the building boundary in anti-clockwise order
3. Load the LIDAR data point from the file
4. Walk through to locate the triangle which contains the LIDAR data point
5. Check if the order of the vertices are anticlockwise
   - Store the point in a list for further calculation if it is true (point A in figure 2), or

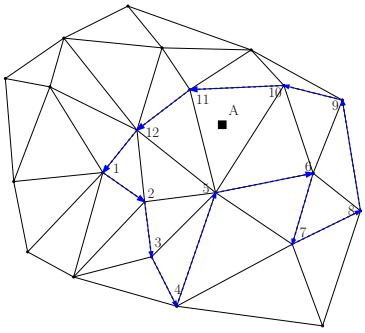


Figure 2: The order of the vertices is anticlockwise 5-10-11

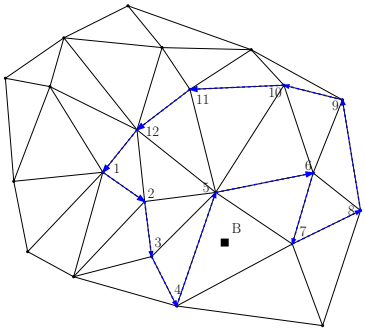


Figure 3: The order of the vertices is clockwise 4-5-7

- Do nothing and go to another point if it is not anti-clockwise (point B in figure 3)

6. Calculate the average height in each list for the building height
7. Repeat from step 3 until all the points are searched

We have added the building boundary to the terrain surface. We will show how to use the Euler Operators to extrude buildings in the next section. Figure 4 shows the building boundaries (top) and the extruded buildings (bottom) of the University of Glamorgan campus.

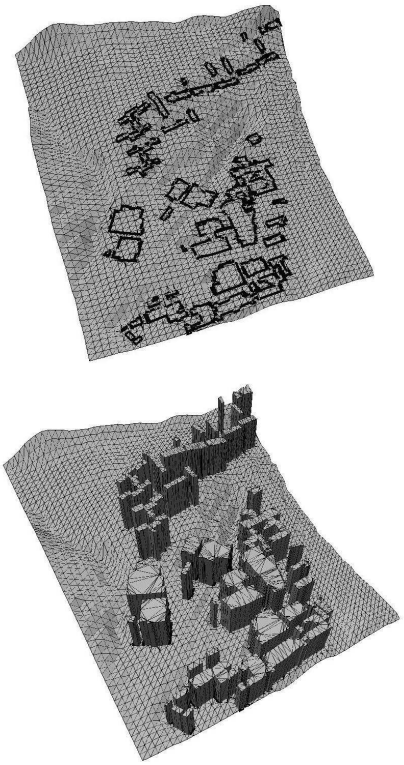


Figure 4: Building boundaries (top) and the extruded buildings (bottom) of the University of Glamorgan campus

2.2 Voronoi City Modelling

Automatic extraction of building boundaries based on edge detection is difficult. Here we superimpose a coarse layer of random Voronoi cells on the LIDAR data and adjust their positions so that each cell is entirely “inside” or “outside” a building. A subset of the cell boundaries then forms the building boundary.

The steps used in Voronoi City Modelling are:

1. Sample the raw LIDAR data to get a lower density of data point and create a Voronoi diagram

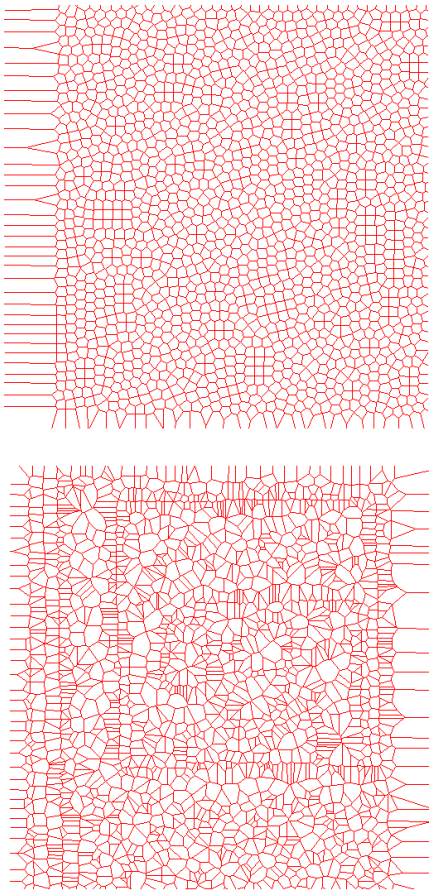


Figure 5: The original (top) and iterated Voronoi Cells in 2D view

from the sampled data points.

2. For each Voronoi cell:
3. Compare the Voronoi cell with its neighbours, and calculate the within-cells and between-cells variance of the underlying LIDAR points. Calculate the F-ratio ("between" / "within").
4. Move the cell and leave it in its new location if the F-ratio increased due to the move.
5. Repeat 2 to 4 until all cells are processed.
6. Iterate steps 1 to 5 until the best building shapes show in the 3D view.

Figures 5 shows the original Voronoi Cells (top) and those after a few iterations (bottom) in a 2D view. Figure 6 shows the 3D view where the building outlines have more or less formed.

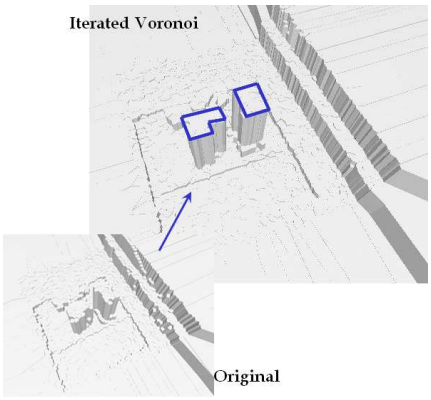


Figure 6: The original and iterated Voronoi Cells in 3D view

3 Building Extrusion Using Euler Operators

Constructive Solid Geometry (CSG) and VRML are the two common methods for modelling and rendering buildings on the terrain. [Bre99] and [SV04] used CSG to generate a complex building with the Boolean operations of union, intersection and differences. [RB03] used VRML to display the generated buildings. However topological relationships are not kept during the construction. All the buildings are superimposed on the terrain surface without actually connecting to it. If the topological connectivity is preserved, more kinds of spatial analysis can be performed.

We start by creating a 2.5D TIN model and used Euler Operators to extend the model. Using Euler Operators to extrude a building has four stages [TG01]:

1. The Basic Quad-Edge data structure (with only two topological operators) [GS85].
2. The implementation of Euler Operators using the Quad-Edge data structure.
3. The implementation of triangulation models using Euler Operators.
4. Interactive building modification using additional Euler Operators.

3.1 The Basic Quad-Edge Data Structure

We used the Quad-Edge data structure as the basis of our model. More particularly, the Quad-Edge structure was used to implement a set of Euler Operators sufficient for the maintenance of surface triangulations. The Quad-Edge data structure fulfills a sufficient adjacency topology representation according

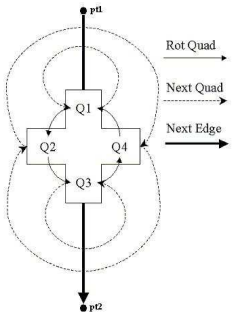


Figure 7: Make Edge creates a new independent edge

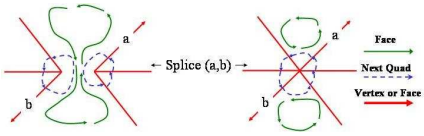


Figure 8: Splice and its own inverse, either splits a face or merges two faces

to Weiler [Wei88]. "Make-Edge" and "Splice" (figures 7 and 8) are the two simple operations on the Quad-Edge structure, which is formed from four connected "Quads" objects, using the simple implementation of [Gol98].

3.2 Implementation of Euler Operators Using the Quad-Edge Data Structure

Five spanning Euler Operators (plus the Euler-Poincaré formula) suffice to specify the number of elements in any boundary representation model [BHS78]. In TINs there are no loops (holes in individual faces), so these will not be considered. Thus four spanning Euler Operators suffice for TINs with tunnels or bridges. We use "Make Edge Vertex Vertex Face Shell" (MEVVFS) to create an initial shell, and its inverse "Kill Edge Vertex Vertex Face Shell" (KEVVFS) removes it. Figure 9 shows the operator MEVVFS and its inverse.

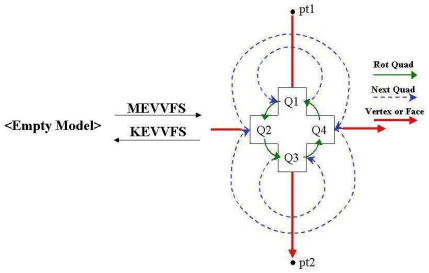


Figure 9: MEVVFS and its inverse KEVVFS

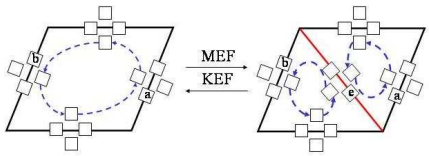


Figure 10: MEF and its inverse KEF

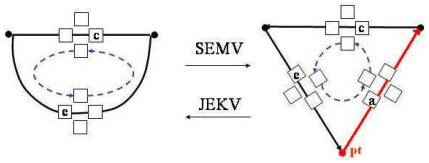


Figure 11: SEMV and its inverse JEKV

In figure 10, "Make Edge Face" (MEF) and its inverse "Kill Edge Face" (KEF) creates and kills an edge and a face. "Split Edge Make Vertex" (SEMV) splits an edge and creates a vertex, and its inverse is "Join Edge Kill Vertex" (JEKV) which show in figure 11

3.3 Implementation of the TIN Model Using Euler Operators

In the TIN model we have three main functions, which are:

- Create the first triangle
- Insert or its inverse delete a point
- Flip an edge

We use MEVVFS, MEF and SEMV to create the first triangle from an empty model in figure 12. The first triangle should be big enough to contain all the data points. JEKV, KEF and KEVVFS are used to kill the first triangle.

Figure 13 shows inserting and its inverse, deleting, a new point which adds three edges and two faces. MEF

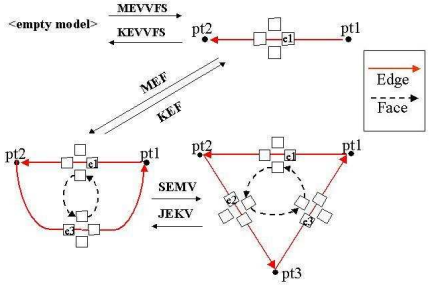


Figure 12: First Triangle should be big enough to contain all data points

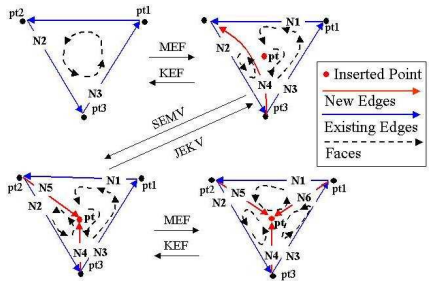


Figure 13: Insert and its inverse delete a point

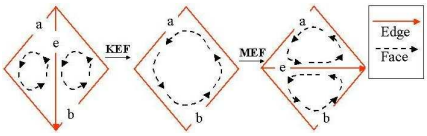


Figure 14: Flip22

creates an edge N4. SEMV splits an edge N4. In the last step MEF creates a new edge N6. We use KEV and JEKV to delete a point. To delete a point, we use MEF to kill an edge and face. Then we use JEKV to join two edges and kill a vertex. We use KEF to kill edge N4 and its associated face.

Flip22 (in figure 14) is a procedure for flipping an edge between two triangles in a TIN. For the Delaunay Triangulation, we use the "in-circle" test to test the triangle, and use the flip22 operator to change edges. The Delaunay Triangulation is based on the empty circumcircle criterion [GS85].

3.4 Building Extrusion Using Euler Operators

We used one more Euler Operator to simplify the procedure. "Make Zero-Length Edge Vertex" (MZEV) and its inverse "Kill Zero-Length Edge Vertex" (KZEV) is used to create and kill a zero length edge and a vertex (figure 15). MZEV can be replaced by using MEF, SEMV and KEF which simplify the whole procedure.

Figures 16 and 17 show extruding a four triangles building from the terrain. We can do more interactive modelling by creating tunnels and bridges from figures 18 and 21

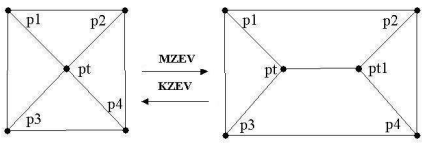


Figure 15: MZEV and its inverse KZEV

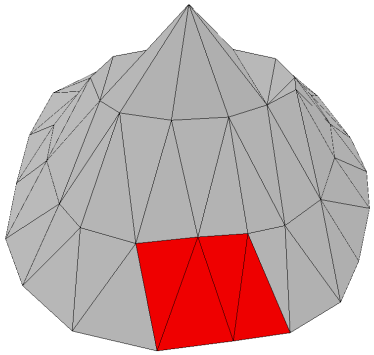


Figure 16: Four triangles are selected to extrude a building

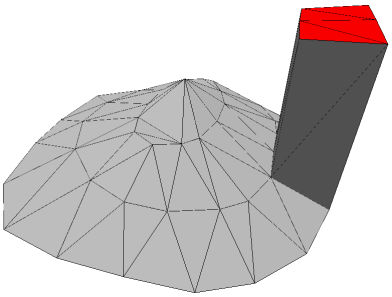


Figure 17: The extruded building

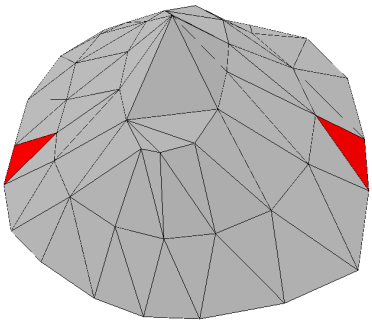


Figure 18: Select two unconnected triangles to create a tunnel

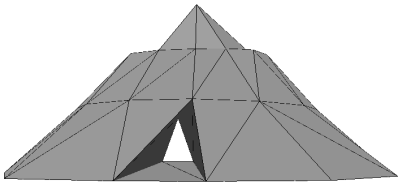


Figure 19: A tunnel is created

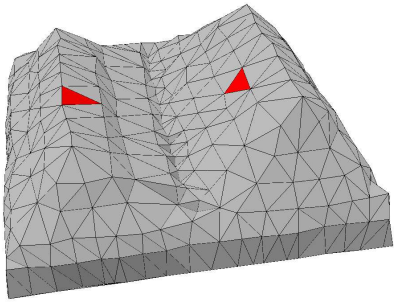


Figure 20: Select two unconnected triangles to create a bridge

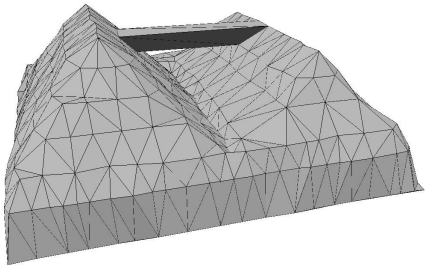


Figure 21: A bridge is created

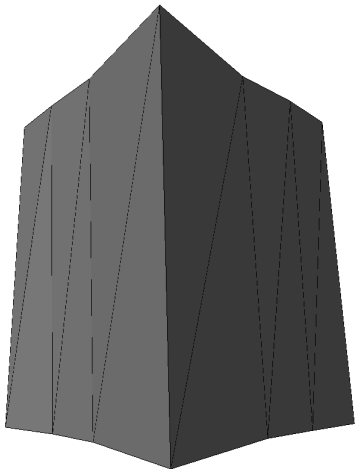


Figure 22: A simple roof

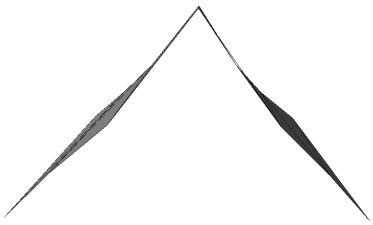


Figure 23: Roof Shape

4 Modelling the Roof Structures

We get extruded buildings on the terrain, but they are all flat-topped. We would like to find a way to model the roof structures. Some can be complicated. Researchers are still looking for a perfect solution for roof reconstruction.

For a simple roof we calculate the eigenvalues and eigenvectors of the vector normals of the triangles which are located inside the building boundary. The smallest eigenvalue gives the orientation of the roof ridge and shows the shape of the roof. Figure 22 shows a 2D view of the roof. The orientation of the smallest eigenvalue gives a view of the roof shape shown in figure 23. For more complex roofs we can plot the vector normals on the unit hemisphere, construct the Delaunay triangulation and then use the minimum spanning tree to extract clusters of points. Each cluster represents one roof face orientation.

5 Conclusions

We have proposed methods for embedding building models within the terrain model, in both the presence and the absence of mapped building boundaries. We have also suggested new roof modelling techniques based on the clustering of triangle vector normal orientations. While we are still refining these methods we believe they address several previously unresolved issues in modelling buildings with LIDAR data.

6 Acknowledgement

The research discussed in this paper was supported by the Ordnance Survey, Southampton (Research and Innovation).

References

[BHS78] I.C. Braid, R.C. Hillyard, and I.A. Stroud. Stepwise construction of polyhedra in geometric modelling. In K.W. Brodlie, editor, *Mathematical Methods in Computer Graphics and Design*, pages 123–141. A Subsidiary of Harcourt Brace Jovanovich, Leicester, 1978.

[Bre99] C. Brenner. Interactive modelling tools for 3D building reconstruction. In D. Fritsch and R. Spiller, editors, *Photogrammetric Week '99'*, pages 23–34, Wchmann Verlag, Heidelberg, 1999.

[DG02] M. Dakowicz and C. M. Gold. Extracting meaningful slopes from terrain contours. In P.M.A. Sloot, C.J.K. Tan, J.J. Dongarra, and

A.G. Hoekstra, editors, *In Proceedings : Computational Science - ICCS 2002, Lecture Notes in Computer Science*, volume 2331, pages 144–153, Amsterdam, The Netherlands, 2002. Springer-Verlag Berlin Heidelberg.

[Gol98] C. M. Gold. The quad-arc data structure. In T.K. Poiker and N.R. Chrisman, editors, *8th International Symposium on Spatial Data Handling*, pages 713–724, Vancouver, BC, Canada, 1998.

[GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[HMS03] A. D. Hofmann, Hans-Gerd Mass, and André Steilein. Derivation of roof types by cluster analysis in parameter spaces of airborne laserscanner point clouds. In H.-G. Maas, G. Vosselman, and A. Streilein, editors, *Proceedings of the ISPRS working group III/3 workshop '3-D reconstruction from airborne laserscanner and InSAR data'*, volume 34, Part 3/W13, Dresden, Germany, 2003. Institute of Photogrammetry and Remote Sensing Dresden University of Technology.

[RB03] F. Rottensteiner and C. Briese. Automatic generation of building models from LIDAR data and the integration of aerial images. In H.-G. Maas, G. Vosselman, and A. Streilein, editors, *Proceedings of the ISPRS working group III/3 workshop '3-D reconstruction from airborne laserscanner and InSAR data'*, volume 34 Session IV, Dresden, Germany, 2003. Institute of Photogrammetry and Remote Sensing Dresden University of Technology.

[Rog02] M. Roggero. Object segmentation with region growing and principal componenet analysis. In *ISPRS commission III Sysmposium "Photogrammetric Computer Vision"*, pages 298–294, Graz, Austria, 2002.

[SV04] I. Suveg and G. Vosselman. Reconstruction of 3D building models from aerial images and maps. *ISPRS Journal of Photogrammetry & Remote Sensing*, 58(3–4):202–224, 2004.

[TG01] R.O.C. Tse and C.M. Gold. Terrain, dinosaurs and cadastres - options for three-dimension modelling. In C. Lemmen and P. van Oosterom, editors, *Proceedings: International Workshop on "3D Cadastres"*, pages 243–257, Delft, The Netherlands, 2001.

[Vos03] G. Vosselman. 3d reconstruction of roads and trees for city modelling. In H.-G. Maas, G. Vosselman, and A. Streilein, editors, *Proceedings of the ISPRS working group III/3 workshop '3-D reconstruction from airborne laserscanner and InSAR data'*, volume 34, Part 3/W13, Dresden, Germany, 2003. Institute of Photogrammetry and Remote Sensing Dresden University of Technology.

[Wei88] K.J. Weiler. Boundary graph operators for non-manifold geometric modeling topology representations. *Geometric Modeling for CAD Applications*, 1988.

Socially Communicative Characters for Interactive Applications

Ali Arya
iMediaTek, Inc.,
Vancouver, BC, Canada
aarya@sfu.ca

Steve DiPaola
Simon Fraser University,
Surrey, BC, Canada
sdipaola@sfu.ca

Lisa Jefferies, James T. Enns
University of British Columbia,
Vancouver, BC, Canada
{ljefferies,jenns}@psych.ubc.ca

ABSTRACT

Modern multimedia presentations are aggregations of objects with different types such as video and audio. Due to the importance of facial actions and expressions in verbal and non-verbal communication, the authors have proposed "face multimedia object" as a new higher-level media type that encapsulates all the requirements of facial animation for a face-based multimedia presentations within one single object. In this paper, Interactive Face Animation - Comprehensive Environment (iFACE) is described as a general-purpose software framework that implements the "face multimedia object" and provides the related functionality and tools for a variety of interactive applications such as games and online services. iFACE exposes programming interfaces and provides authoring and scripting tools to design a face object, define its behaviours, and animate it through static or interactive situations. The framework is based on four parameterized spaces of Geometry, Mood, Personality, and Knowledge that together form the appearance and behaviour of the face. iFACE can function as a common "face engine" for design and run-time environments to simplify the work of content and software developers.

Keywords

face animation, parameter, behavior, personality, geometry, game, interactive

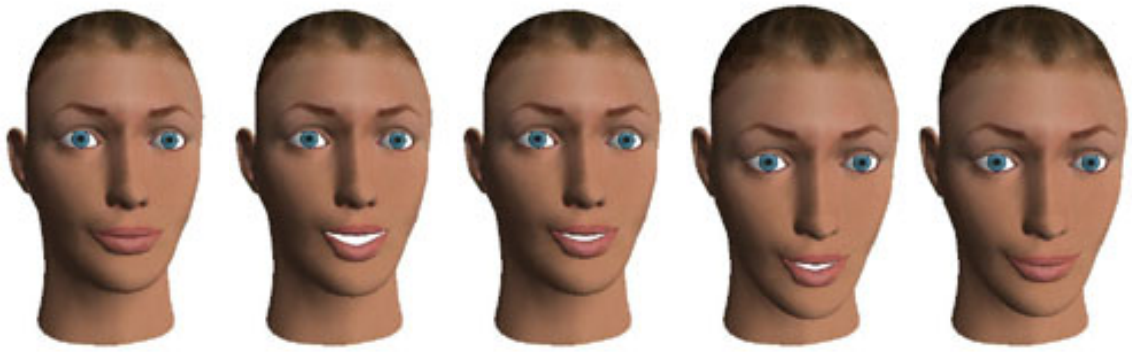


Figure 1. Sample animated 3D heads showing expressions, talking, and moving head

1. INTRODUCTION

Advances in computer hardware and software have introduced the Interactive Multimedia Presentation as a common base for a variety of audio-visual applications and computer-generated facial animation is a rapidly growing part of such presentations. For instance, although current computer games make limited use of facial expressions, next generation game platforms provide hardware capabilities for computations involved in having more degrees of freedom in characters.

One of the main objectives of game designers is to utilize these new platforms to introduce more realistic characters who can change expressions more frequently, demonstrate personality traits more clearly, and behave more interactively. A virtual customer service representative can be considered another application of such characters.

Some of the issues facing content and application developers of realistic interactive characters are:

Behaviour. Designing different facial actions, expressions, and personality traits usually involve a painstaking and time-consuming process where

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

artists create the related animation using conventional 3D software and by defining key frames for the movement of each facial feature.

Re-usability. Designs for one head model are not generally usable on another model. As a result, even a similar action on a new head requires the design process to be repeated.

Interaction. The need for a detailed design process limits the amount of interactivity and dynamic behaviour a character can have at run-time. In other terms, the characters can not be completely autonomous.

Programmability. There is a serious lack of programmable components that can be re-used in new applications to provide facial animation capabilities.

Level of Details. The animators, especially when using conventional graphics software, have to deal with all the details of a head model to perform actions. An intelligent software that is aware of head regions and their function can hide the details unless necessary, by performing group actions on all the points that are functionally related.

In this paper, we introduce Interactive Face Animation – Comprehensive Environment (iFACE) that provides solutions to all of the above problems in a unified face animation framework. iFACE parameter spaces allow animator and/or programmer to effectively control facial geometry, perform MPEG-4 compatible facial actions, show expressions, and display behaviours based on definable personality types. All of these are encapsulated within a Face Multimedia Object (FMO) that can be used in any applications through programming interfaces.

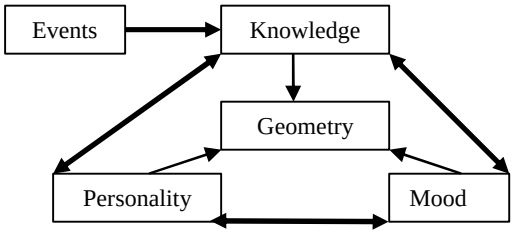


Figure 2. iFACE Parameter Spaces

iFACE hierarchical geometry arranges parameters in different layers of abstraction to allow exposure to proper level of details. The physical points can be pixel or vertex to support 2D and 3D models, both with the same control interface. On top of them are the feature points which correspond to MPEG-4 Face Animation and Definition Parameters [Bat99]. Using only these parameters almost any facial action is possible. Features and component layers are at higher levels and allow grouping of functionally related parameters. The parameters and their respective API are independent of the head model, so a set of parameters can be applied to any head model, resulting in the same facial actions.

Dynamic behaviours in iFACE are possible through Knowledge, Mood, and Personality parameter spaces. They allow defining interaction rule and scripts written in Face Modeling Language (FML), expressions, and personality types. iFACE personality types are based on the state-of-the-art in behavioural psychology. They are defined as a combination of Affiliation and Dominance factors and control the way facial actions are performed (e.g. frequency of blinking, typical head movements, etc). Using these parameters, an autonomous character can be created which interacts properly in a dynamic environment.

The parameterized approach with its behavioural extensions allow the animators and runtime programmers to use “face” as a self-supporting object without the need for dealing with details, and apply the same parametric design to any head model. iFACE API on the other hand, provides a powerful flexible component-based structure to be used in any application that requires face animation.

Some related works in face animation are briefly reviewed in Section 2. The main concepts of FMO and iFACE framework are discussed in Section 3. iFACE software architecture is also briefly introduced in this section. In section 4, a few applications are presented that use iFACE. Some concluding remarks and discussions are the subject of Section 5.

2. RELATED WORK

The common practice for face animation is to use general-purpose 3D modeling and animation tools such as Alias Maya, Discreet 3DS Max or SoftImage XSI. While providing very powerful design environments, these tools lack dedicated face-centric features that allow efficient and realistic modeling and animation of facial states and actions [Ekm78]. The run-time environments, consequently, animate the limited degrees of freedom provided by the authoring tools, and do not support any face-specific run-time support (such as dynamic creation of typical behaviours) that can simplify application development. Although this situation might be marginally sufficient for current level of face animation in games, and match the computational power of existing game consoles, it is far less than ideal, especially for next generation games running on much more powerful platforms.

One of the earliest works on parameterized head models was done by Parke [Par72]. The parameterized models are effective ways due to use

of limited parameters, associated to main facial feature points, compared to complicated simulation of physical entities [Par00]. Facial Action Coding System (FACS) [Ekm78] was an early and still valid study of possible facial actions related to such feature points. Although not originally a computer graphics technique, FACS has been widely used by researchers in parameterized models and others. This approach has been formalized in MPEG-4 standard by introduction of Face Definition Parameters and Animation Parameters (FDP,FAP) [Bat99, Ost98].

The primary issue with such parameter space is its "flatness". All parameters are worked with in a similar way while not every application actually needs all of them. A hierarchical grouping of parameters is necessary for efficient management of parameter space. Also, the relatively huge amount of parameters (result of extensions to the original models) makes application development and authoring hard. A common solution to this issue has been defining higher-level parameters and behaviors [Byu02, Cas94, Dec02]. For example, "smile" is an action that involves a few features and can be defined at a higher level of abstraction, knowing the combined effect of movements in feature points. MPEG-4 FAPs define two groups of such high level parameters for standard facial expressions and visemes (visual representation of uttered phonemes).

Although such "macro" parameters make it easier to use the underlying model, the simple two-tier model is still not very effective for managing facial activities and providing local control over level-of-details. The MPEG-4 standard allows definition of parameter groups but it is only a standard to be used by particular models, which are still mainly "flat". Fidaleo and Neumann [Fid02] propose using regions to group points with similar movements. Pasquariello, and Pelachaud [Pas01] (among others) have proposed hierarchical head models that allow a more efficient parameter control through grouping and regions. Our approach, as explained later, uses this idea and extends it to multiple layers of abstraction on top of actual data points (2D pixels or 3D vertices) to ensure maximum flexibility and minimum effort when group actions are required. Our head model pyramid has a head object on top, and components, features, feature points, and physical points are at lower levels.

Finally, the behavioral modeling of animated characters has been studied by some researchers. Funge et al. [Fun99], for instance, define a hierarchy of parameters. At the base of their parameter pyramid is the geometric group. On top of that come kinematic, physical, behavioral, and cognitive parameters and models. Although very important for introduction of behavioral and cognitive modeling concepts, the model may not be very suitable for face animation purposes due to the interaction of parameter groups and the need for emotional parameters as opposed to physically-based ones.

Cassell et al. [Cas94, Cas01] defined behavioral rules to be used in creating character actions but do not propose a general head model integrating geometrical and behavioral aspects. Byun and Badler [Byu02] propose the FacEMOTE system that allows four high-level "behavioural" parameters (Flow, Time, Weight, and Space) to control the expressiveness of an input FAP stream. Although it demonstrates how high-level behavioural parameters can control facial animation, it does not intend to be a comprehensive face object. On the other hand, three spaces of Knowledge, Mood, and Personality (each with their own parameters as explained later) can control the facial behaviour in a more explicit way. RUTH system by DeCarlo et al. [Dec02] uses behavioural rules to animate a face when a given text is spoken. Smid et al. [Smi04] use a similar approach but associate a considerably larger set of facial actions (head, eye, brow movements) to features of a given speech through behavioural rules in order to create an autonomous speaker agent. Although these rules can be base for defining personality types, the possibility has not been explored by these researchers. Pelachaud and Bilvi [Pel03] propose performative dimensions (dominance and orientation) and emotion dimensions (valence and time) as behavioural parameters to control facial actions. The systems share common concepts but iFACE provides a more comprehensive framework for defining personality types and custom expressions, and it is based on studies in behavioural psychology to associate facial actions to these personality types and expressions. Also, iFACE allows interactive non-verbal scenarios through an XML-based scripting language, MPEG-4 compatibility at lower levels, multimedia streaming, authoring tools, programming interfaces, and wrapper applets for form-based applications. Personality space also allows a more general mechanism for defining facial personalities.

3. IFACE SYSTEM

Face Multimedia Object

The ability to create a multimedia presentation as a combination of parts with different types has resulted in new multimedia standards such as MPEG-4. The simplest of these types can be audio and video. The need for more efficient multimedia authoring and management suggest that such object-based approach be extended to more "aggregation" in multimedia content, i.e. grouping of related content elements into higher-level "types". For a variety of cases where human figures play a key role ("face-centric" applications) "face" is a primary candidate for such a

data type. The introduction of Face Definition and Animation Parameters (FDPs and FAPs) in MPEG-4 standard was a step toward such higher-level of abstraction on top of face-related multimedia content. The authors have proposed Face Multimedia Object (FMO) [Ary04] as a more systematic approach to encapsulate face functionality into one autonomous but controllable object.

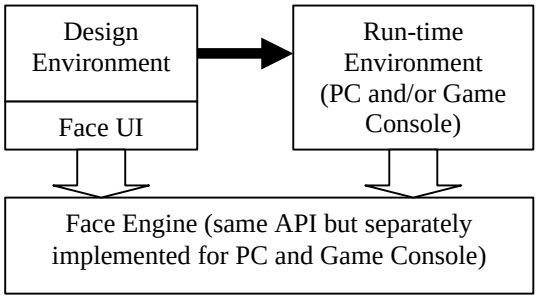


Figure 3. Using Face Multimedia Object

As shown in Figure 3, FMO operates as a "face engine" for design-time and run-time applications. Using FMO, animators and authors can design proper geometry and facial actions and pass them to run-time modules only as commands, instead of keyframes with information on all moving parameters. At run-time, the application/game only detects the required action and asks the engine to replicate the same result. This has the advantages such as:

- Less information saved by design tool and passed to run-time
- Ease of run-time development due to black-box use of FMO
- Possibility of dynamic applications and user-controlled event-driven scenarios without the need of a pre-design

Parameter Spaces

Rousseau and Hayes-Roth [Rou97] consider Personality Traits, Moods, and Attitudes as major parameters in their social-psychological avatar model. In a similar but revised way, we believe that the communicative behavior of a face can be considered to be determined by the following parameter spaces:

- *Geometry*: This forms the underlying physical appearance of the face that can be defined using 2D and/or 3D data (i.e. pixels and vertices). This geometry is based on a hierarchy of facial components, features and points as illustrated in Figure 4.
- *Knowledge*: Behavioral rules, stimulus-response association, and required actions are encapsulated into Knowledge. In the simplest case, this can be the sequence of actions that a face animation character has to follow. In more complicated cases, knowledge can be all the behavioral rules that an interactive character learns and uses.
- *Personality*: Different characters can learn and have the same knowledge, but their actions, and the way they are performed, can still be different depending on individual interests, priorities, and characteristics. Personality encapsulates all the long-term modes of behavior and characteristics of an individual [Wig88, Bor92]. Facial personality is parameterized based on typical head movements, blinking, raising eye-brows and similar facial actions.
- *Mood*: Certain individual characteristics are transient results of external events and physical situation and needs. These emotions (e.g. happiness and sadness) and sensations (e.g. fatigue) may not last for a long time, but will have considerable effect on the behavior. Mood of a person can even overcome his/her personality for a short period of time. Emotional state can be modeled as point in a 2D space where two axes correspond to energy and value [Rus80].

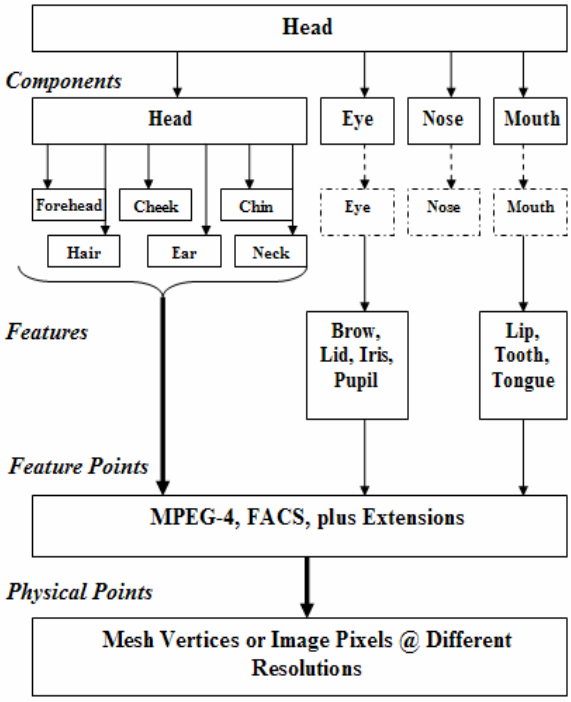


Figure 4. iFACE Geometry Hierarchical Model

Face Geometry

As explained in Section 2, hierarchical geometry in iFACE provides a more efficient way of accessing parameters. Geometry Components allow grouping of head data into parts that perform specific actions together (e.g. resizing the ears or closing the eye). Features are special lines/areas that lead facial

actions, and Feature Points (corresponding to MPEG-4 parameters) are control points located on Features. Only the lowest level (Physical Point) depends on the actual (2D or 3D) data. iFACE Geometry object model corresponds to this hierarchy and exposes proper interfaces and parameters for client programs to access only the required details for each action. iFACE authoring tool (iFaceStudio) allow users to select Feature Points and Regions, i.e. groups of points that usually move wit a feature point (similar to co-articulation regions by Fidaleo and Neumann [Fid02]). Each level of Geometry accesses the lower levels internally, hiding the details from users and programmers. Regions can be defined by selecting 3D point using a 3D editor, or defining an area on 2D texture. iFACE head model includes 60 Regions.

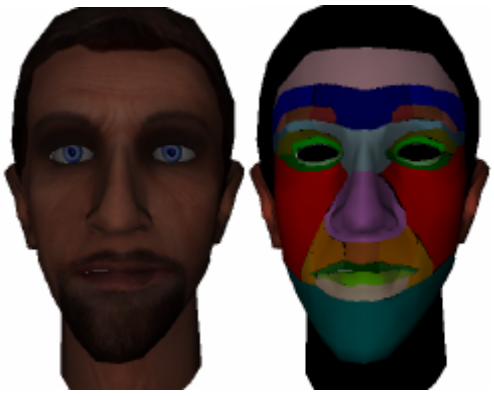


Figure 5. Face Regions for a 3D head. These are small areas that usually move together and are controlled by Feature Points. iFACE Components are related groups of these Regions, e.g. eye area

Eventually, all the facial actions are performed by applying MPEG-4 FAPs to the face. FAPs themselves move feature points and regions. Each FAP has a maximum displacement vector (normalized in FAP units) and an activation value. Together these two define the movement of the corresponding feature point due that FAP. Other Region points follow this feature point by distance-dependent weight function as illustrated in Figure 6. The slope and half-weight distance on this function are configurable. The movement of each point is the sum of all movements related to active FAPs (Better methods are being studies).

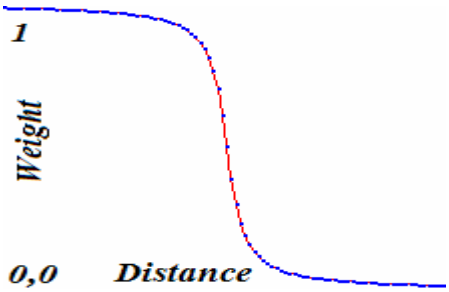


Figure 6. Weight of movement as a function of distance from feature point

Face Modeling Language

The behaviour of an iFACE character is determined primarily by Knowledge. It provides the scenario that the character has go through as an XML-based script. iFACE uses Face Modeling Language (FML) [Ary04] that is specifically designed for face animation. FML document can be a simple set of sequential actions such as speaking and moving the head, or a complicated scenario involving parallel actions and event-based decision-making similar to the following script:

```
<fml>
  <model><event name="kbd" /></model>
  <story>
    <action>
      <!--parallel actions-->
      <par>
        <hdmv type="yaw" value="80"
            begin="0" end="2000" />
        <play file="Audio1.wav" />
      </par>
      <!--exclusive actions -->
      <!—-depending on event value-->
      <!--one of options will run-->
      <excl ev_name="kbd">
        <talk ev_value="F1_down">
          Hello</talk>
        <talk ev_value="F2_down">
          Bye</talk>
      </excl>
    </action>
  </story>
</fml>
```

iFACE Knowledge module exposes interfaces to allow opening new scripts or running single FML commands. It also allows defining and raising program-controlled events that are base for dynamic and interactive scenarios.

Mood

Although scripts can select a new personality or modify the mood, but Knowledge space is generally independent of the "character". Mood and Personality spaces deal with character-dependent parameters. Mood controls short-term emotional state that can affect the way a certain action is animated. For instance actions in a part of script can be performed in a "happy" mood and in another part in a "sad" one and be visually different. In general, moods are represented with a certain facial expression with which any facial action is modulated (i.e. overlaid on top of actions, Figure 7), as follows:

$ACT = sum(FAP_1, FAP_2, \ldots, FAP_n)$
$EXP = sum(FAP_1, FAP_2, \ldots, FAP_m)$
$DISPLAY = f(BASE,EXP)$

In these equations, *ACT* is the facial action to be performed consisting of *n* FAPs, *EXP* is the current mood made of *m* FAPs, and *DISPLAY* is the final

face output. *f* is currently a simple addition but more complicated functions are being investigated.

(a) (b)

Figure 7. Talking for a Cartoonish 3D Head in a Neutral (a) and Happy (b) Mood

iFACE supports two types of moods each with a zero to one activation level:

- Standard emotions (joy, sadness, surprise, anger, fear, disgust) [Ekm78]
- Custom expressions defined by user

It is also possible to select the current mood of character by adjusting Energy and Stress values which will result in activation of standard emotions at some level according to Russell's Circumplex mood model [Rus80] as shown in Figure 8 where horizontal and vertical dimensions are Stress and Energy, respectively.

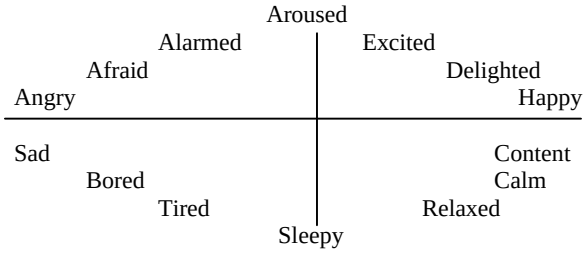


Figure 8. Russell's Circumplex Mood Model

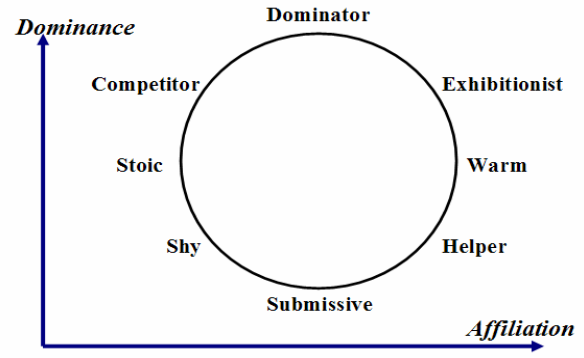


Figure 9. Personality Circumplex Model

Face Personality

Interpersonal Adjective Scale [Wig88] is a widely accepted personality model that links different personality types to two Affiliation and Dominance parameters in a two dimensional Circumplex model (Figure 9). Facial actions and expressions are shown to cause perception of certain personality traits [Bor92, Knu96].

The foundation of iFACE personality is associating major facial actions and expressions with personality parameters and types, i.e. visual cues for personality. This is done based on published works and our own on-going research (described in Section 4 as an iFACE application). When the personality parameters are changed or a certain personality type is activated, the associated facial actions are selected to be performed (i.e. visual cues are presented) in order to make perception of that personality type more probable in the viewer. Following personality-related actions are defined:

- Expressions
- 3D head movements
- Nodding
- Raising/lowering/squeezing eyebrows
- Gaze shift
- Blinking

For each one of personality-related facial actions, strength, duration, and frequency of occurrence are controllable. The visual cues can happen randomly, with a pre-defined order, or based on the voice energy when talking. Two threshold values are set for speech energy: Impulse and Emphasis. When the energy reaches any on of these thresholds, certain visual cues of current personality type are activated, for instance nodding when emphasizing on a part of speech. We use ETCodec lip-sync module by OnLive that calculates a speech energy for each audio frame of 60mSec. ETCodec also gives values for mouth shape which are translated to MPEG-4 FAPs by iFACE. Sample associations between visual cues and perceived personality types are shown in Table 1. These are the results of an on-going study which is the subject of another paper.

Visual Cue	Perceived Personality Type
Happiness and surprise	high in dominance and affiliation
Anger	high in dominance and low in affiliation
Sadness and fear	low in dominance
Averted gaze	low affiliation
Looking away (turning)	low affiliation
Raising head	high dominance
One-sided eyebrow raise	high dominance
Tilted head	high affiliation
Lowering head	Low dominance
Frequent blinking	low dominance

Table 1. Visual Cues and Personality Types

Software Architecture

iFACE is developed as a set of .NET components written with C#. It is possible to access them directly from .NET Managed Code or through .NET/COM

Interop from Unmanaged Code. Code written for .NET framework (e.g. a C# program) is called Managed Code. Normal Windows and Component Object Model (COM) code are considered Unmanaged. .NET allows interoperability with COM objects using a mechanism called COM Interop. iFACE uses Microsoft Direct3D and DirectSound for graphics and audio purposes. In cases where Unmanaged Code was required (for instance using existing ETCodec lip-sync library) a COM object is developed to wrap the code and use it in iFACE. Implementation of iFACE FMO on game consoles as a possible run-time environment, and iFACE plug-in components for Maya and 3DS-MAX are on-going projects. iFACE is designed to work with a variety of client types. Depending on the level of details exposed by the components, iFACE objects are grouped into three layers:

- Data: main objects such as Geometry and Behaviour
- Stream: extra objects for streaming
- Wrapper: form controls to simplify the use of face functions

Current version of iFACE supports only 3D head models and animation. Full support for the hierarchical head model on top of 2D data is under development. The head models are initially created using traditional 3D software and then can be customized using iFaceStudio software which also helps define the regions and components, and configure parameters (e.g. FAP maximum displacements), and also define FAP combinations corresponding to higher-level parameters. iFaceStudio can also be used to create animation files with FML commands or key-frames defined by parameter values.

4. EXPERIMENTAL APPLICATIONS WITH IFACE

Face Personality Study

Behavioural psychology researchers usually use photographs and less commonly video to perform experiments. They can benefit from an interactive environment that can create realistic animations with different features (e.g. mood and personality). This can replace actors which are hard or expensive to find with software that does not need external setup and can be easily configured. iFACE system is being used in such an application which in turn provides information regarding how viewers perceive the personality of a subject based on his/her facial actions.

Using iFACE the researcher can change the personality traits (or any other aspect) of the subject and observe the reaction and perception of the viewers. For more information see the web site: http://ivizlab.sfu.ca/arya/Research/FacePersonality

COMPS

Simulation and Advanced Gaming Environment for Learning (SAGE, http://www.sageforlearning.ca) initiative is a joint project among Simon Fraser University and some other Canadian universities. Collaborative Online Multimedia Problem-based Simulation Software (COMPS) is a system being developed by SAGE to support online Problem-based Learning (PBL) for medical students. PBL works by introducing students to a case (problem), giving them some facts, and taking them through cycles of discussion and hypothesizing until the learning objectives have been met. iFACE is used in COMPS to represent a remote instructor and simulate patients.

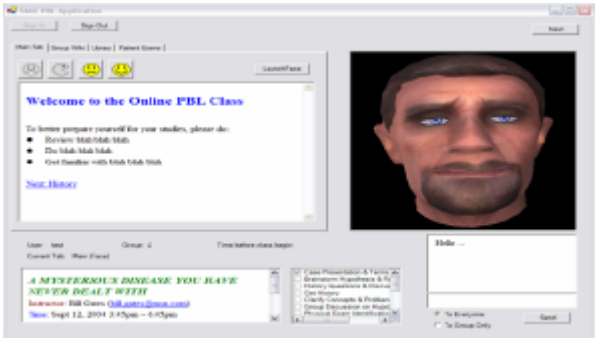


Figure 10. COMPS with a Simulated Patient

Storytelling Masks

iFACE is used in a museum environment to create animations of native North American artists explaining their work and the myths related to them, as illustrated in Figure 11.

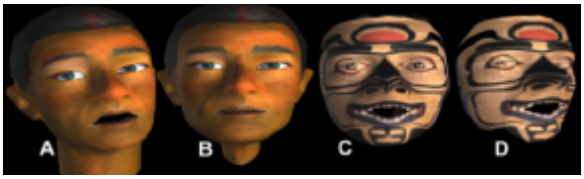


Figure 11. Frames from Storytelling Masks

Evolving Faces

Human migration, as explained in “out of Africa” theory, is illustrated in this application using talking faces of each region/age, as shown in Figure 12.

Figure 12. Screenshot of Evolving Faces

5. CONCLUSION

In this paper, we introduce the iFACE as a framework for face multimedia object. iFACE encapsulates all the functionality required for face animation into a single object with proper application programming interface, scripting language, and authoring tools. iFACE use a hierarchical head model that hides the modeling details and allows group functions to be performed more efficiently. Multiple layers of abstraction on top of actual head data make the client objects and users independent of data type (3D or 2D) and provide the similar behaviour regardless of that type.

Behavioural extensions in form of Knowledge, Personality, and Mood control scenario-based and individual-based temporal appearance of the animated character. On the other hand, streaming and wrapper objects make the use of iFACE components easier in a variety of applications. iFACE framework is a powerful "face engine" for character-based online services, games, and any other "face-centric" system.

iFACE is fully compatible with MPEG-4 standard. Although higher level parameters exist, they are all translated to MPEG-4 FAPs before being performed. The system uses a dedicated XML-based language which provides all necessary scripting functionality in addition to normal XML Document Object Model (DOM).

Future research on iFACE will involve comprehensive association of all facial actions and expressions to most likely personality type to be perceived, exploring the possibility of higher level parameters in face personality (on top of affiliation and dominance) in order to define practical character types such as nervous and heroic), and realistic combination of current mood and facial actions by using non-linear functions. More information on iFACE system can be found at:
http://ivizlab.sfu.ca/research/iface
http://www.imediatek.com

7. REFERENCES

[Ary04] Arya, A., and DiPaola, S. 2004. Face As A Multimedia Object, In *Proceedings of WIAMIS*, Lisbon, Portugal, April 21-23.

[Bat99] Battista, S., et al. 1999. MPEG-4: A Multimedia Standard for the Third Millennium, *Multimedia*, vol. 6, no. 4, IEEE Press.

[Bor92] Borkenau, P., and Liebler, A., 1992. Trait Inferences: Sources of Validity at Zero Acquaitance. In *Journal of Personality and Social Psychology*. Vol 62. no 4, pp 645-657.

[Byu02] Byun, M., and Badler, N.I., 2002. FacEMOTE: Qualitative Parametric Modifiers for Facial Animations. In *Proceedings of ACM SIGGRAPH/ Eurographics symposium on Computer Animation*, ACM Press.

[Cas94] Cassell, J., et al. 1994. Animated Conversation: Rule-based Generation of Facial Expression, Gesture and Spoken Intonation for Multiple Conversational Agents.. In *Proceedings of ACM SIGGRAPH*, ACM Press.

[Cas01] Cassell, J., et al. 2001. BEAT: The Behavior Expression Animation Toolkit. In *Proceedings of ACM SIGGRAPH*, ACM Press.

[Dec02] Decarlo, D., et al. 2002. Making Discourse Visible: Coding and Animating Conversational Facial Displays. In *Proceedings of Computer Animation Conference*, IEEE Press.

[Ekm78] Ekman, P. , and Friesen, W. V. 1978. *Facial Action Coding System*, Consulting Psychologists Press Inc.

[Fid02] Douglas Fidaleo, D., and Neumann, U., 2002. CoArt: Co-articulation Region Analysis for Control of 2D Characters, In *Proceedings of Computer Animation*.

[Fun99] Funge, J. et al. 1999. Cognitive Modeling. In *Proceedings of ACM SIGGRAPH*, ACM Press.

[Knu96] Knutson, B. 1996. Facial Expressions of Emotion Influence Interpersonal Trait Inferences, *Journal of Nonverbal Behavior*, 20, 165-182.

[Ost98] Ostermann, J. 1998. Animation of Synthetic Faces in MPEG-4. In *Proceedings of Computer Animation Conference*, IEEE Press, 49–55.

[Par72] Parke, F. I. 1972. Computer Generated Animation of Faces. In *Proceedings of ACM Annual Conference*, ACM Press, 451–457.

[Par00] Parke, F. I., and Waters, K. 2000. *Computer Facial Animation*. A. K. Peters.

[Pas01] Pasquariello, D., and Pelachaud, A., 2001. Greta: A Simple Facial Animation Engine. In *6th Online World Conference on Soft Computing in Industrial Applications*, Session on Soft Computing for Intelligent 3D Agents.

[Pel03] Pelachaud, W. and Bilvi, M. 2003. Computational Model of Believable Conversational Agents. In *Communication in MAS: Background, Current Trends and Futur*e, Marc-Philippe Huget (Ed), Springer-Verlag.

[Rou97] Rousseau, D., and Hayes-roth, B. 1997. Interacting with Personality-Rich Characters. *Knowledge Systems Laboratory Report No. KSL 97-06*, Stanford University, September 1997.

[Rus80] Russell, J.A., 1980. A Circumplex Model of Affect. In *Journal of Personality and Social Psychology*, 39, 1161-1178.

[Smi04] Smid, K. et al. 2004. Autonomous Speaker Agent. In *Proceedings of Computer Animation and Social Agents*, Geneva, Switzerland.

[Wig88] Wiggins, J.S. et al. 1988. Psychometric and Geometric Characteristics of R-IAS, *Multivariate Behavioural Research*, 23, 517-530

Granular Material Interactive Manipulation: Touching Sand with Haptic Feedback

Bedřich Beneš
Computer Graphics Technology
Purdue University
bbenes@purdue.edu

Enkhtuvshin Dorjgotov
Computer Graphics Technology,
Purdue University
edorjgo@purdue.edu

Laura Arns
Envision Center,
Purdue University
larns@purdue.edu

Gary Bertoline
Envision Center,
Purdue University
bertoline@purdue.edu

Abstract

We present a novel approach for a virtual granular material interactive manipulation with force feedback. A user can interactively change a height-field model of sand by dragging objects inside. The virtual sand behaves like real sand moving to the sides and falling back, filling holes and irregularities on the surface. The dragging object position is controlled by a haptic device that provides a position in 3D space that correspondingly changes the sand model. The sand model provides force feedback to the haptic device resulting in two principal forces; the repulse that has vertical direction and the viscous drag. The resulting force is delivered back to the haptic device. The user can sense the sand's response as the dragging object moves through the virtual sand. This results in the haptic-visual feedback providing a higher degree of plausibility than visual feedback alone.

Keywords: Sand, Tactile Devices, Virtual Reality, Simulating Natural Phenomena.

1 INTRODUCTION

Terrain modeling algorithms and techniques published in the area of computer graphics such as [1, 2, 3, 5, 6, 11, 10, 14] have focused mostly on obtaining visually realistic physically-based models. Little attention has been paid to real-time and interactive techniques for producing realistic shapes of terrains. Interactive manipulation techniques, such as [7, 8, 9, 13] mostly focus reaction of sand or soils to a certain user action. One of the first commercially successful applications providing interactive terrain modeling was the Metacreations Bryce 3D®.

We present a new algorithm for interactive modeling of sand shape using haptic devices. A user "touches" the virtual sand, which changes the shape correspondingly (see Figure 1). The potential users and applications of this algorithm are anyone that needs to rapidly and realistically model the shape of sand. There are many interactive modelers for different data types. For example, there is the NURBS sculptor in MAYA®, hair painter in the same software, particles can be modeled in 3D MAX® and MAYA®, etc. The authors are not aware of any application that allows interactive modeling of sand.

The paper begins with a description of the previously published modeling and simulating techniques that work with sand in the area of computer graphics. In Section 3 we describe the real-time modeling and displaying of sand, and Section 4 describes the haptic device and the force feedback from the sand. Section 5 discusses the implementation issues and the last part shows results of our experiments and concludes the paper.

WSCG 2006 conference proceedings, ISBN 80-86943-03-8
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

Figure 1: WSCG written in sand

2 RELATED WORK

The previous work can be roughly divided into two significant classes: the real-time algorithms and the non-real-time algorithms. We will first discuss the historically older group of non-real-time algorithms.

2.1 Non-Real-time Sand Manipulation

Probably the first paper dealing with sand simulation is the erosion simulation algorithm of Musgrave et al [10]. Material is described as a regular height field and is eroded by thermal and hydraulic erosion. Some parts are broken by the thermal shocks and fall down. Some amount of material is dissolved by running water and deposited to a different location. The shape morphology is described as a set of parameters that defines the material properties.

Sumner et al [13] introduced a technique for animating sand and soil motion that results from interactions

with tools. Their technique is able to simulate material displacement that results from footsteps and trails in the sand and mud.

An interesting approach for modeling wind ripples in sand was shown by Onoue and Nishita [7]. The sand particles reallocation is formed by the two principal factors - creep and saltation. These phenomena are described by a set of equations that is applied to the sand model represented as a regular height field.

Beneš and Forsbach [1] introduced a layered data structure that allows for representation of volumetric effects, such as caves, as well as for a surface erosion. They use a RLE-like data structure that represents layers of deposited material. This volumetric technique reports comparable speed of erosion simulation as in the case of height-fields that is significantly better than voxel-based approaches.

Zhu and Birdson [14] recently presented application of the Navier-Stokes equations to simulation of sand motion. Sand is represented as a cloud of particles and its motion is described by the physically correct equations. Special attention is paid to the surface tracking and rendering.

2.2 Real-time Sand Manipulation

The real-time manipulation algorithms are represented by the following previous work.

Li and Moshell in [9] described a physically-based model of real-time digging and caving. The model works with extended regular height-fields and describes the surface changes.

The papers [1, 13] were the motivation for the Virtual Sandbox of Onoue and Nishita [8]. Real-time manipulation of sand is described by the dual sand representation; as a height field and as a set of particles. Particles are used for sand that is elevated over the surface of the height-field and the height-field is used for deposited sand. The authors report interactive frame rates for sand manipulation and interaction.

Neidhold et al [11] recently introduced another algorithm for terrain shape morphology that deals with homogenous sand-like structures and allows for real-time hydraulic erosion simulation.

All of the previous work focuses on shape morphology and the interaction is limited to the user working with a mouse as an input device or to real-time displaying of erosion processes. We propose a haptic-device interaction that provides another level of fidelity to users who interact with the granular material model.

3 REAL-TIME SAND SIMULATION

One of the principal advantages of sand, from the viewpoint of modeling in computer graphics, is the fact that it cannot form concave structures. That is why sand is usually represented as a regular height-field; a two dimensional matrix, where each vertex corresponds to the elevation of the given location. The elevation points are regularly sampled in two dimensional space, which simplifies algorithms for the terrain metamorphosis simulation. On the other hand this representation is space demanding. Regardless of the level of detail or the amount of terrain roughness the amount of space occupied is the same. We use regular height-fields to store the model of the sand.

There are two aspects of the sand manipulation to be considered. The sand relocation by user action (*displacement*) and sand returning to the stable position by gravity (*erosion*). Our algorithm is a two pass algorithm. In the first pass the user contacts the sand with a virtual tool and the dragging displaces a certain amount of the material. In the second pass the sand is eroded back. We will describe both passes in depth.

3.1 Sand displacement

As an object penetrates the sand surface it displaces a certain amount of the granular material. This displacement depends on the force vector that represents the drag direction, the collision area between the object, and on the object and the sand boundary shape. The force is responsible for the depth the objects enter the granular material, and its shape determines the amount of pushed sand. General calculation can be very demanding, but careful scene preparation can simplify it.

We use a sphere as the dragging object. It has the principal advantage of having the same shape facing the sand regardless of dragging direction. The amount of displaced material can be easily determined.

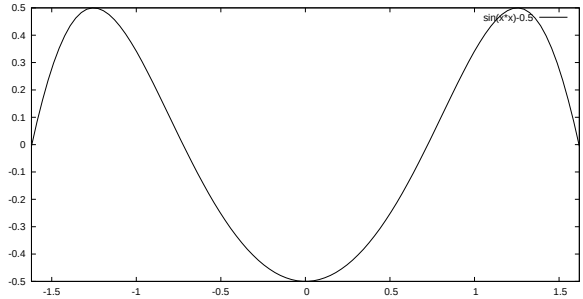


Figure 2: The cross-section of the filter applied to replace the sand

To make the calculation even faster we introduce a concept of two two dimensional erosion discrete filters. The first one has circular shape and corresponds to the area that penetrates the sand and all values in this filter are negative. We denote this filter by $neg_{i,j}$. The other filter has positive values, has a shape of circular rampart, and is located around the negative filter. It is denoted by $pos_{i,j}$. The sum of the integrals over both

filters is zero which reflects the volume preservation condition.

$$\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} neg_{i,j} + \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} pos_{i,j} = 0.$$

Both filters are displayed in one image in Figure 2. The inner part begins in the inflexion points of the curve and it is the *neg* filter. The rest is the outer part i.e., the *pos* filter. The discrete filters result from sampling a continuous function that has the form

$$neg_{x,y} = sin(x^2+y^2) - 0.53, \quad \sqrt{x^2+y^2} > 1.2,$$

$$pos_{x,y} = \begin{cases} sin(x^2+y^2) - 0.53 & 1.63 < \sqrt{x^2+y^2} \leq 1.2, \\ 0 & otherwise \end{cases}$$

When the dragged object penetrates the sand we first apply the circular negative filter. Based on the depth of the impact, the filter values are shifted by a constant coefficient and the sum of the values from this filter and the terrain is calculated. The result of this operation is one number which is the amount of the material that the sphere pushes out. At the same time, the negative values are added to the terrain that removes the material.

The removed material is distributed around the object according to the values in the positive filter in the second pass. The second pass is applied in the same manner as the first one i.e., applying the filter to the area around the object. The only difference is that the values from the filter are multiplied by the proportional part of the material detected in the first pass. In this way only the material detected in the first step is removed in the second one and that is why we need the two step algorithm. The requirement of the material reallocation is essential. The result of this operation is displayed in an example in the upper image of the Figure 3

3.2 Sand erosion

Sand seeks gravitational equilibrium. Any particle that is placed in a higher position tends to fall down and find the lowest possible point. This process corresponds to the thermal weathering and has been described by Musgrave et al [10] and later used by various authors [2, 8, 13]. The result of application of this technique is shown in the lower image of the Figure 3. We will briefly describe the simulation algorithm here.

Each point of the height field is processed. The actual height is compared with its eight neighbors and, if it exceeds at least one of them, the corresponding part of sand is removed down. Material is transported only to the neighboring elements so several passes are required to reach equilibrium.

Let's denote the height of the processed vertex by m and the height of the neighbors is m_i, $i = 1,2,\ldots,8$.

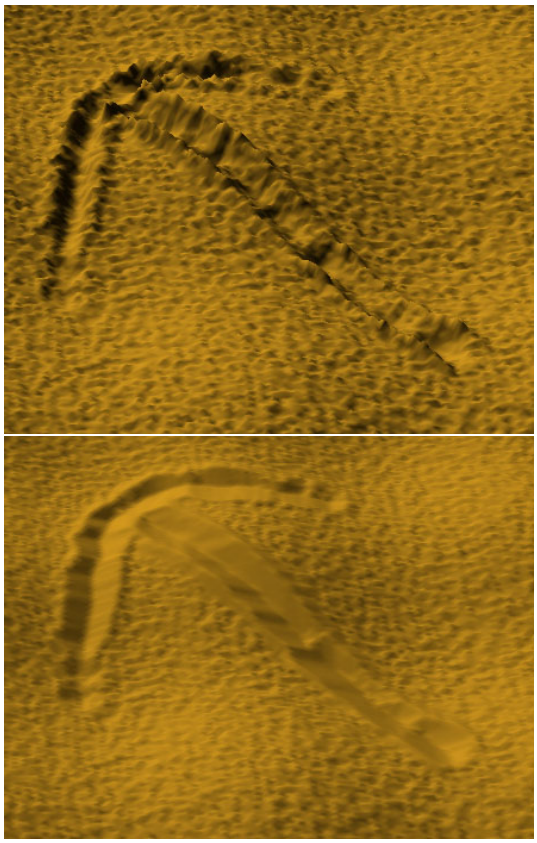


Figure 3: Example of the application of the positive and negative filters (up) and the same scene after erosion

There are two possible cases: $m < m_i; \forall i = 1,2,\ldots,8$ and the opposite one.

The first case corresponds to a vertex being a local minimum so there is no material that can be removed from it. In the second case at least one vertex lies lower than the actually processed vertex and the corresponding part of material is removed down. To get the correct distribution of the transposed sand, we first calculate the amount of material that can be reallocated. It is then redistributed according to the height of its neighbors. The vertices that are located lower than the others will receive a higher portion of the material. The total amount of the sand that is relocated is denoted by Δm and the redistribution is described by

$$\Delta m_i = \Delta m \frac{m_i}{sum}.$$

In this equation Δm_i is the sand that is moved to the i-th lower located vertex, *sum* is the sum of all differences to the lower located vertices, and m_i is the actual difference to the i-th neighbor.

There is an additional condition of the so called talus angle [10]. This condition postulates that sand does not move to the direction of a lower located neighboring vertex if the slope is smaller than the talus angle. This phenomenon reflects the inner tension of the granular material and roughly says that no erosion goes to infin-

ity. Depending on material, equilibrium is reached at a faster or slower pace. Especially in the case of sand, the talus angle is found very quick, as everyone who has visited the beach has probably experienced. When digging a hole in the sand, it is filled very fast but only the hole boundary is softened. This is shown in the Figure 4, where a scene was set up in such a way to demonstrate the phenomenon.

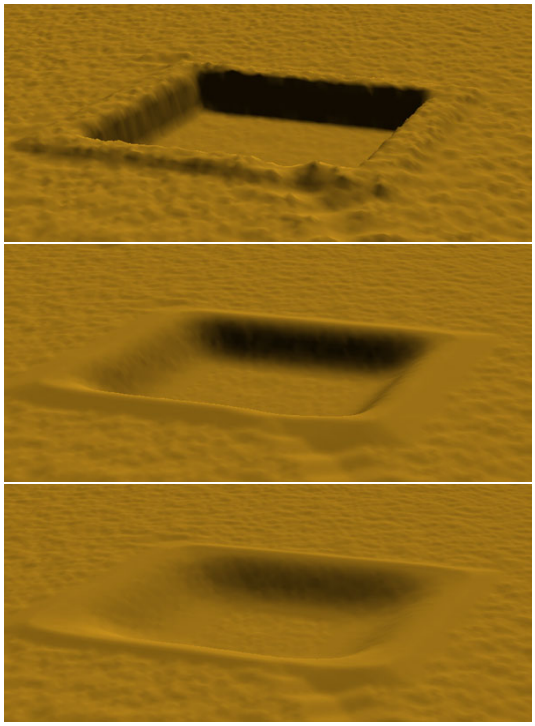


Figure 4: Sand finds the talus angle fast

Setting the value of the talus angle is essential for the speed of erosion. Erosion weakens exponentially and it does not make any sense to keep the angle close to zero. A small talus angle only keeps the algorithm running without a significant visual effect. On the other hand high talus angle values causes the granular material to look very rough. We have experienced a reasonable value of the talus angle to be around 30^o.

4 TOUCHING THE SAND

The fundamentally new approach in this paper is using haptic devices to interact with a material that is a subject of metamorphosis at the same time.

Sand is a special case of a granular material and is an extremely complicated subject that is studied in physics. This is primarily because the forces inside the material are not well-known yet. Granular material can be considered as a tight particle system. Each particle has its stickiness, surface area, mass, and shape. As an external force is applied the particles on the surface rotate and move and the forces are transferred inside the volume. This presents a vast amount of forces that are distributed inside the granular material. Obviously, the force distribution depends heavily on the size of the particles and their stickiness (humidity is an important factor here). The above described qualitative model can be substituted by a quantitative one using reasonable simplifications.

4.1 Forces

We propose a model of sand interaction to a force under the following conditions [12]. The penetrating object is a sphere. It has the major advantage that its interaction area is the same regardless of the direction the sphere moves. This allows us to drop-off area-to-sand calculations and this increases the speed of simulation. Sand is homogenous in our calculations. There are no layers with higher humidity, there are no crunchy shells on the surface, no cracks, nor bubbles or stones. Based on these assumptions we set up a set of equations that describe the force feedback of the sand to the penetrating object. The forces are then sent to the input of the haptic device which provides the corresponding feedback.

The resulting force has two important components. The vertical *repulse force* and the horizontal *friction* as can be seen in Figure 5.

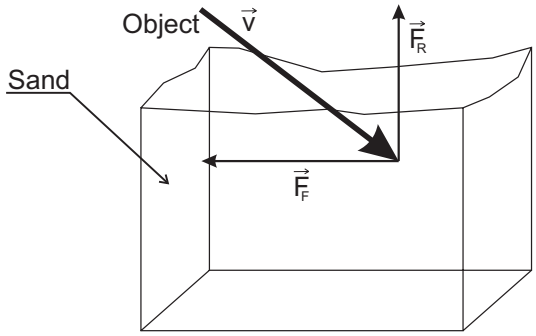


Figure 5: An object entering the sand is repulsed by the vertical force and dragged by friction

The *repulse force* results from the increasing tension with the depth of the sand. The repulse force has the opposite direction to the normal vector to the surface, close to the surface, and becomes vertical with increasing depth. The repulse function has the form

$$\vec{F_r} = -k_s e^{k_c d} - 1 \qquad (1)$$

where k_s, k_c are the material dependent constants and d is the depth of the layer of sand. This force has always vertical, or nearly vertical direction. We use the strictly vertical direction in our simulations, since the sand erodes quickly to the talus angle, as described in Section 3. The angle is small so the normal vector to the surface is nearly vertical. The difference is impossible to feel.

The second force is the inner *friction* of the sand that has only the vertical component. It is also a function of depth but it is higher with the velocity. The equation of friction is

$$\vec{F_f} = -k\vec{F}|\vec{F_r}|. \qquad (2)$$

The force $\vec{F}$ depends on the friction area, the velocity $\vec{v}$, and the size of the repulse force $\vec{F}_r$. The friction area is constant so $\vec{F} = k_a\vec{v}$. We can accumulate the constants $k_f = k\,k_a$ obtaining

$$\vec{F}_f = -k_f\vec{v}|\vec{F}_r|. \quad (3)$$

The three constants depend on the implementation and the relative sizes of the other objects in the scene. Having the total sand area of size $\langle -1,1\rangle^2$ and the sphere radius equal to 0.01, the $k_s = 10^3$, $k_c = 1/2$ and $k_f = 10$.

Figure 6 demonstrates the use of force on one stroke. The stroke starts with a strong vertical force that weakens. This causes more material to be removed at the beginning. The sand after it is touched by the sphere immediately erodes-out the differences.

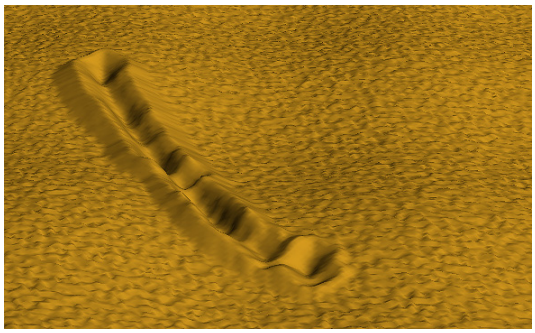


Figure 6: An example of the stroke with varying force applied. The stroke starts with a stronger force and ends with a smaller one as can be seen by the amount of the removed material

4.2 Haptic Devices

Figure 7: The 6DOF Phantom (left) and the 3DOF-Omega haptic device

Our sand drawing application currently works with the 3-DOF Omega(tm) haptic device and 6-DOF Phantom(tm) Desktop haptic device (Figure7), both provided by The Force Dimension(tm). These two haptic devices provide an affordable desktop solution that is suitable for our application. The Omega(tm) haptic device is connected to a PC by the USB port and provides resolution of 0.009mm and maximum force feedback of 12N. The Phantom(tm) is connected via the parallel port and provides 0.03mm resolution, but goes up to 20N and has six degrees of freedom. Both devices provide full gravity compensation.

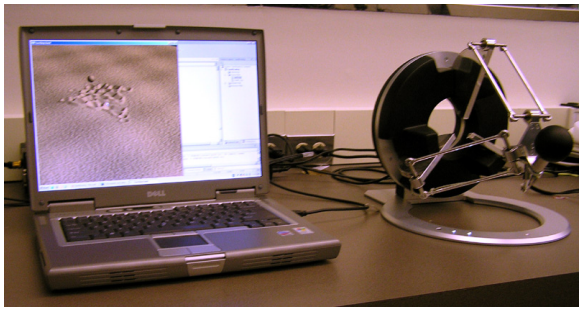


Figure 8: Scene setup

5 IMPLEMENTATION

The entire system is implemented in C++. We use OpenGL(r) for visualization and the CHAI 3D open-source haptic rendering library [4] for interaction. We have chosen the CHAI 3D library because of its transparent support for several haptic devices including the Omega(tm) family, the Delta(tm), and the Phantom(tm) devices. This library is provided in source code and is easily extendible for other haptic devices.

The sand drawing application consists of two major parts; the graphics (visualization) part and the haptic interactive device. Both parts work together to imitate drawing on sand with haptic feedback.

In each haptic rendering loop, we read the actual position of the haptic device in 3D space. We transform it as a position of the virtual device and if it is inside the sand surface we compute the responding forces. These forces are applied to the sand surface and it is indicated that the area should be eroded. The erosion step, that is actually slow, takes place outside the main haptic loop.

The update rates of these two loops are very different since a visual rendering needs a much slower refresh rate compared to a haptic update rate. While a 60Hz refresh rate is standard for visual rendering, the haptic rendering requires at least a 1kHz update rate to provide smooth haptic feedback to a human user. In our application, we use a high frequency timer for the haptic feedback so that the haptic rendering loop takes place in every millisecond. The commands in the haptic loop are not buffered. Once entered the callback application does not execute the same routine until it is terminated. If there is some piece of code that takes a long time, the newly generated position can be much farther. The haptic loop is no reentrant. It is important to keep all the complicated calculations out of the haptic loop and use it just for setting some variables. Actually, synchronization of the visual and the haptic loops presents an interesting problem in general.

Slowing down the application while inside the callback of the haptic loop is especially difficult when changing the depth of the virtual pointer. The force in a shallow depth is small and when deeper it is much higher. According to equation (1) the feedback is an exponential function of depth. Wrongly implemented haptic feedback can cause jumps and sudden abrupt forces. A very strong vertical force on a slow feedback could move the virtual pointer too deep into the granulate material. The haptic device produces a very strong back force in such cases.

There are not major issues in the rendering loop. One of the important parts, from the viewpoint of speed, is to update continuously only the areas that are really changed. To achieve this we divide the regular height field area into a set of 10x10 OpenGL display lists. Once the user makes a stroke, only the affected display lists are updated. Once the area is eroded (all the angles in the simulated area are smaller than the talus angle) and no material is moved, the display lists are not updated anymore. Technically this means that the system is slowest when there are many areas eroded (touched) at the same time. This is technically difficult, because the erosion reaches equilibrium fast.

6 CLOSING REMARKS

The system provides interactive feedback for height-fields up to 1000x1000 vertices (2MΔ). The system runs at 60fps on a Dell Precision M70 laptop running at 1.6GHz with the NVIDIA Quadro FX Go 1440 graphics card.

The haptic feedback does provide a higher level of fidelity than using only visual feedback. This can be easily tested by disabling the force feedback on the haptic device and manipulating the sand directly by mouse. We have tried a mouse but found the haptic feedback gives much better results.

The sequence of images in Figures 9-12 shows the successive modeling of a granular surface. The images are frames from the accompanying video. These images show the result of successive editing of granular material using our system.

We have demonstrated that implementation of interactive sand sculpting system with haptic feedback is possible and leads to better results than just visual feedback. We believe that future applications of our approach could be as a plug-in for some professional systems such as Maya(r), or 3D Studio MAX(r). We believe it would be much better to model complex scenes by simple haptic feedback and the user could then test complicated animation sequences in this way as well.

There are still many open areas and future work should focus on the following unsolved problems:

- dragging arbitrary shapes and correct repulse calculation based on the actual intersection area,

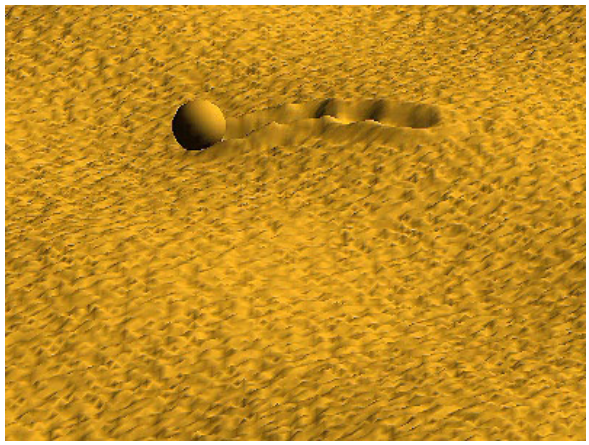


Figure 9: Sequence of images demonstrating successive editing of a granular material (image 1)

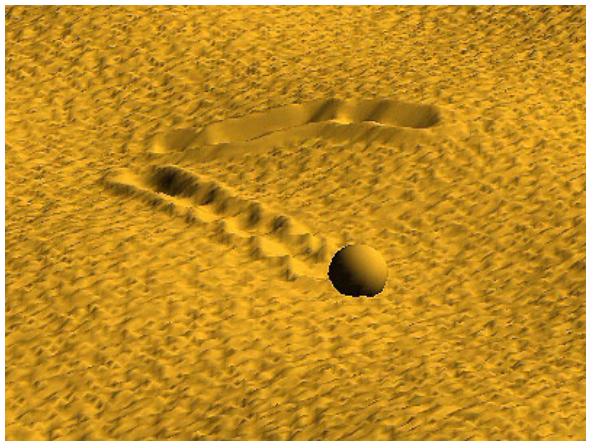


Figure 10: image 2

- correct material manipulation,
- including non-homogenous surfaces, shells, stones, wet sand, and
- including sand in the air (for example as free particles as done in the work of Onoue and Nishita [8]) and

Figure 11: image 3

Figure 12: image 4

- improving control over the haptic device and the rendering loop.

There is one last interesting observation. The haptic feedback is incomparably better than just a visual feedback as we tested when just using a mouse instead of the Phantom(tm) device. On the other hand, using the haptic devices intensively, for more than four hours in the case of the final tuning the application, causes *very strong pain* in the arm next day. This is certainly an aspect that should be considered. The forces that a user has to support are up to 3 N, which means manipulating 3kg of material with one hand. Doing this for a long period of time is certainly good exercise for the user.

REFERENCES

[1] B. Beneš and R. Forsbach. Layered Data Structure for Visual Simulation of Terrain Erosion. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, volume 25(4), pages 80–86. IEEE Computer Society, 2001.

[2] B. Beneš and R. Forsbach. Visual simulation of hydraulic erosion. *Journal of WSCG*, 10(1):79–86, 2002.

[3] N. Chiba, K. Muraoka, and K. Fujita. An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and Computer Animation*, 9:185–194, 1998.

[4] F. Conti, F. Barbagli, R. Balaniuk, M. Halg, C. Lu, and D. Morris. The CHAI libraries. In *Proceedings of Eurohaptics 2003*, pages 496–500, Dublin, IE, July 6–9 2003.

[5] J. Dorsey, A. Edelman, H. W. Jensen, and H. .K Pedersen. Modeling and Rendering of Weathered Stone. In *Proceedings of SIGGRAPH '99*, volume 25(4) of *Computer Graphics Proceedings, Annual Conference Series*, pages 225–234. ACM, ACM Press / ACM SIGGRAPH, 1999.

[6] T. Ito, T. Fujimoto, K. Muraoka, and N. Chiba. Modeling rocky scenery taking into account joints. In *Computer Graphics International*, pages 244–247, 2003.

[7] O. Koichi and T. Nishita. A Method for Modeling and Rendering Dunes with Wind-ripples. In *Proceedings of Pacific Graphics'00*, pages 427–428, 2000.

[8] O .Koichi and Tomoyuki. Nishita. Virtual sandbox. In *Proceedings of Pacific Graphics'03*, pages 252–260. IEEE Computer Society, 2003.

[9] X. Li and M. Moshell. Modeling Soil: Realtime Dynamic Models for Soil Slippage and Manipulation. In *Proceedings of SIGGRAPH'93*, volume 27(4) of *Annual Conference Series*, pages 361–368, 1993.

[10] F.K. Musgrave, and C.E. Kolb, and R.S. Mace. The Synthesis and Rendering of Eroded Fractal Terrains. In *Proceedings of Siggraph'89*, volume 23(3) of *Annual Conference Series*, pages 44–50, 1989.

[11] B. Neidhold, and M. Wacker, and O. Deussen Interactive physically based Fluid and Erosion Simulation. In *Proceedings of Eurographics Workshop on Natural Phenomena*, volume 1, pages 25–32, 2005.

[12] G.N. Smith and G.N. Ian Smith. *Elements of Soil Mechanics*. Blackwell Science Profesional, 1998.

[13] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins. Animating Sand, Mud, and Snow. *Computer Graphics Forum*, 18(1):17–26, 1999.

[14] Y .Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.

Motion Edit with Collision Avoidance

Li Liu
[1] Institution of Computing Technology, Chinese Academy of Sciences
[2] Graduat School of Chinese Academy of Sciences
Beijing China, 1000801
liuli@ict.ac.cn

Zhao-qi Wang
Institution of Computing Technology, Chinese Academy of Sciences
Beijing China, 1000801
Zqwang@ict.ac.cn

Zhu Deng-ming
[1] Institution of Computing Technology, Chinese Academy of Sciences
[2] Graduat School of Chinese Academy of Sciences
Beijing China, 1000801
dmzhui@ict.ac.cn

Shi-Hong Xia
Institution of Computing Technology, Chinese Academy of Sciences
Beijing China, 1000801
xsh@ict.ac.cn

ABSTRACT

The existing motion editing methods don't take collision between the limbs into account potentially producing implausible motions. Facing this problem, in this paper we present an integrated framework of motion editing for producing collision-free motion in real-time. We first provide an efficient scheme for collision detection based on skeleton model, which can fast find the penetration between limbs of human body. Then we provide a novel scheme for constraint generation, which embeds the motion characteristics and inter-frame coherency into the constraint generator, and preserves the qualities of the original motion and motion consistency in the target motion. Finally, we show how to apply the Kalman filter to constraint resolve in a real-time manner. Comparing to other existing methods, our method not only can obtain the collision-free motion in real time, but also can preserve the original characteristics as many as possible and generate the new motion similar to the original motion. Experiment shows that our approach is very useful in producing natural and collision-free motion and efficient enough for application in animation system and game.

Keywords

motion editing; self-collision detection; collision avoidance; constraint-based animation

1. INTRODUCTION

Motion capture technique is an increasing popular approach for generating realistic human motion. However, the captured data is only for special person in performing specific motion. Whenever the model or the environment is changed, the data must be edited account for different anthropometric scales or reaction to environment interactions.

In the past few years, researchers have developed various constraint-based motion editing techniques [Tak05, Gle98, Lee99, Shi01, Cho00]. These techniques mainly focus on deforming the captured data to satisfy the given constraints and generating new natural-looking motion. But they seldom consider the potential collision between limbs of the articulated figure. Behavioral reasonable properties may therefore be lost in the editing process, making motions appear unrealistic. For example, if we simply transfer walk motion of a thin character to a fat man, the hand will likely to interpenetrate the body. So, detection and avoidance self-collision are vital for the human-like animation.

Although the collision-free is so important in motion editing, unfortunately, producing collision-free motion is still a difficult task. Firstly, collision detection algorithm can determine if the interpenetration has occurred in the editing process. Virtual human model is typically represented by thousands of meshes. Since these meshes need to be updated and overlapped test according to the movement of the human body at each frame, the cost of this computation is huge. Secondly, in addition to fast collision detection algorithm is required, a real-time constraint resolve algorithm is also necessary to further accelerate the computational speed. Finally, as we know that the motion editing methods are based on the reference motion, thus it is important to preserve the original motion characteristics in the

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

editing process without extra computation cost. Unfortunately, current motion editing techniques are largely difficult to satisfy the last two requires at the same time.

Overcoming the challenges outlined above, combining with the techniques of collision detection, constraint generation and constraint resolve, we, in this paper, present a novel approach to generate self-collision-free motion in real time.

The main contributions of our approach as follows:

(1) We select Oriented Bounding Boxes (OBBs) as the coarse hierarchy representation of human body. For the human model, the OBB can easily be constructed and updated. Moreover the OBB can bound the geometry tightly and efficiently eliminate the meshes that are absolute collision-free within a frame.

(2) We provide a fast collision detection method. Unlike the previous methods, we integrated the limit of kinematic reachability for joint into the collision detection algorithm. The pair of bounding boxes that can not close each other won't be checked. Thus, the number of pairs to be detected will substantially reduce. In addition, the method can be applied in other research fields, such as the robotics.

(3) Applying the kalman filter to per-frame resolve the constraints. Different from the previous method, kalman filter not only can accurately deal with the nonlinearity of constraints, but also can minimize the difference between before and after the motion editing. Thus the pose at each time frame can be optimal estimated and the characteristics of the captured motion can be well preserved in the final motion.

(4) Presenting a constraint generation technique. Per-frame method requires acquire the end-effector kinematics constraints for each frame in advance, However the existing methods of constraint generation, such as key frame interpolation [Bin98] and reusing the end-effector position of the performer to the target character [Cho00, Bad93], only can guarantee the interaction between the target character and the environment, the motion characteristics of original motion will be lost. For this reason, we present a reasonable and efficient method of constraint generation by integrating the captured motion characteristics and inter-frame coherence into the constraint generator.

The rest of this paper is organized into several sections: We begin our discussion with related work in section 2, and then provide an overview of our scheme in section 3. Technique of constraint generation is provided in section 4. In section 5, kalman filter is applied to per-frame resolve the constraints in real time. In section 6 we describe our method of self-collision in detail. We demonstrate some experimental results in Section 7. Finally we conclude the paper and discuss the future work in section 8.

2. RELATED WORK

2.1 Collision Avoidance

To our knowledge there is little work in human motion animation that edits captured motion considering the collision avoidance. However, many algorithms in other research fields have been designed with the same goal.

In the robotics field, collision avoidance and path planning are very crucial techniques, and many exact and heuristic solutions have been developed [Can88]. With the increasing number of the DOF (Degrees Of Freedom), however, the computing complexity of these methods grows exponentially. Obviously, these solutions can not satisfy the requirement of collision avoidance in human animation.

In the physically based animation field, collision was detected and dynamic equations were applied to simulate collision response [Ban95]. However, in human motion animation, when some key frames are changed to avoid collision, the whole duration of the motion, rather than local, should be modified correspondingly. So for the human motion animation, instant reaction is not a good choice.

In the inverse kinematics field [Hua96, Zha94], they use the sensors to detect the collision, and collision avoidance is integrated into the inverse kinematics equation. But the method is progressive: each time a collision is detected, the inverse kinematics brings the end-effector to a particular place. Thus, the motion goes from one place to another place in the end, which does not ensure a coherent or realistic motion.

Close to our problem is the work of J-C.Nebel[Neb00], who was interested in generating collision-free motion with key-frame interpolation. After collision has been detected, his method creates new sub-key frames between the given key frames. Then these key and sub-key frames are interpolated to create new collision-free motion. However, the key-frame interpolation may severely lose the original characteristics of captured motion, thus makes very hard to generate a new motion similar to the captured motion. Facing this problem, we present an integrated framework of motion editing for generate collision-free motion. Comparing with Nebel's work, our method can well preserve the characteristics of the captured motion. We also

develop a new self-collision detection algorithm special for the articulated model in our method, which can significantly accelerate the detection rate.

2.2 Motion Editing

Motion editing has been well-studied in the past few years and some of commonly used methods are based on spacetime optimization and per-frame methods. Gleicher [Gle98] presented a method using the spacetime optimization technique within the whole motion duration. In his method, inter-frame cohenrence can well be preserved in the target motion, but the computation is huge and only can deal with short motion sequence at interactive speed. For the per-frame method, it can handle infinitely long motion sequence in real time, but the inter-frame cohenrence is difficult to be guaranteed in the editing process. Lee and Shin [Lee99] divided the motion editing problem into per-frame inverse kinematics following by B-Spline fitting. Tak[Tak05] applied kalman filter and least-squares filter. Both techniques use per-frame method with a post filter to smooth the motion. But the two phases will interact in some cases. For example, post filter will ruin the kinematics constraints done by per-frame resolve. Recently, Choi and Ko [Cho00] developed an on-line retargeting method based on per-frame inverse rate control, Shin et al [Shi01] also presented an on-line technique based on IK and the notion of dynamic importance of end-effectors. Without the post filter, the two methods maintain the inter-frame coherence by inputting the continuous path. However the existing per-frame techniques do not introduce an acceptable method to generate constraint trajectory, which not only can satisfied specified constraints, but also can contain the captured motion characteristics. For this problem, we provide an elegant solution to develop a reasonable constraint generator.

2.3 Collision Detection

Current approaches to collision detection seldom deal with self-collision detection of chain structure, but rather propose general methods. These approaches are usually divided into two main classes: one is based on features, and the other is based on bounding volume (BV) and spatial decomposition techniques. The feature-based approaches [Van97, Bro01] exploit the spatial and temporal coherence and compute minimal separation distance. However they are only suited for the case of simple convex polyhedron. Therefore, the method can not handle self-collision detection of human body. Human body is a very complex polyhedron containing thousands of meshes. Typical examples of BV include axis aligned bounding boxes (AABB) [Van97] and oriented bounding boxes (OBB) [Got96b]. Hierarchical structure include sphere [Qiu94], cone tree, k-d tree and octrees [Sam89], other special representation also include binary space partitions (BSP) [Nat90] etc. Although these methods can be used to detect self-collision of human body, but that is very wasteful. According to analyses above, we apply the OBB hierarchy to represent human body, combine the separate axis theorem and the method of triangle-triangle test to detect self-collision, together with the topology information and joint reachability limit to avoid checking collision-free limb pairs.

3. OVERVIEW

In this study, we present a method to generate realistic and collision-free motion in real time. The method combines our previous work [Liu05] with collision detection technique. Figure 1 demonstrates the flow chat of our method. The constraint generator is first applied to generate constrains, which satisfy the given constraints. Then kalman filter is used to edit the captured motion according to the given constraints. Next the edited motion is sent to a collision avoidance system and self-collision between limbs is detected. If there is no collision, which means the result is correctly. Otherwise, based on the detected result, we interactively correct some key frames to avoid collision in the next editing step. That is to say, adding new constraints to the original motion. Finally the corrected key-frame is fed back to constraint generator to reconstruct a new constraint trajectory for the end-effector of human body. The procedure will iterate until the final motion is collision-free and satisfies the specified constraints.

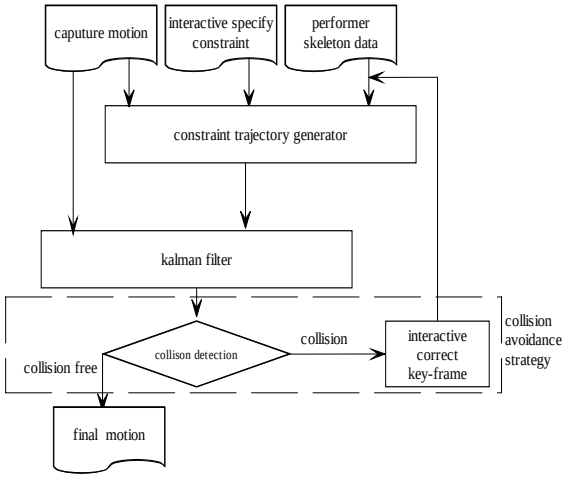


Figure 1.The flow chart of motion editing with collision avoidance

4. CONSTRAINT GENERATOR

Constraints are some features that are to be preserved or changed in the target motion. They contain three kinds of data: the constraint type, the scope of the constraint and constraint value. Specification of

constraint is an important step, because constraints will directly control the properties or details of the target motion. For example, if we specify the restriction of elbow in advance, it won't bend backwards or obtain unrealistic target motion.

Our constraint resolver is based on per-frame kalman filter approach. Since per-frame method need specify constraints at each frame, we can write the constraint as $M(t)$, $t \in [t_0, t_1]$, where $M(t)$ is the constraint value at time *t*, $[t_0, t_1]$ is the scope of the constraint. For the different type constraint, $M(t)$ has different expression.

4.1 Kinematics Constraint

As we stated earlier, our motion editing method is based on reference motion. So original motion pattern should be well preserved in the target motion, otherwise the existence of the reference motion has no sense. Up to this point, we not only need consider satisfying the given spacial constraints, but also must integrate the characteristics of captured motion into constructing kinematics constraint. (See detail in [Liu05])

How to contain the motion characteristics into the kinematic constraint is a great challenge. In our method, we solve it from the following two steps:

(1) Since the joint angle data of captured motion contains some important motion features, and the forward kinematic expression can restore such features effectively, we construct a base end-effector constraint trajectory by entering the captured data q^{org} and skeleton information of target character l^{target} into the forward kinematic equation f_{fk}.

$$M^{org}(t) = f_{fk}(q^{org}, l^{target}) \quad t \in [t_0, t_1]$$

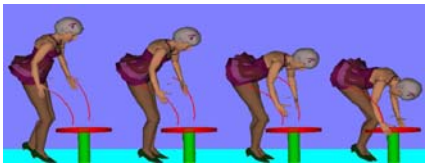
Although captured motion characteristics have been well preserved in this trajectory $M^{org}(t)$, it still can not completely satisfy the given constraints. In the next step, we will further modify the base constraint trajectory according to the given constraints; moreover, we should avoid losing motion characteristics as few as possible.

(2) How to modify the constraint trajectory is another important problem. The Result of the motion frequency analysis [Unu95] has proved that low frequency components of a motion stand for the basic ingredient of a motion, such as motion amplitude, while high frequency components denote the motion details. Consequently, in order to preserve the original motion characteristics as many as possible, we apply the technique of displacement mapping to construct kinematic constraint trajecotry, thus will avoid adding new high frequency information to the target motion. So the final constraint expression can be written as

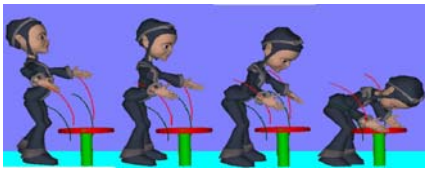
$$M(t) = M^{org}(t) + d(t) \qquad t \in [t_0, t_1]$$

Where $d(t)$ is a displacement trajectory, which can be obtained by linear or cubic interpolation based on the displacement value d_k. The d_k can be computed by subtracting the $M^{org}(t_k)$ from $c(t_k)$ at time k, that is: $d_k = c(t_k) - M^{org}(t_k)$

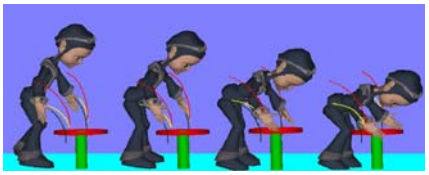
(a) Two different character models



(b) The original motion of bending over



(c) The edited result based on the end-effector trajectory of the performer



(d) The edited result based on our constraint generator

Figure 2: Compare the edited result based on the different constraint trajectory

Figure 2 proves the advantage of our method. The red curve is the end-effector of performer, the black curve shows the trajectory of the motion that is generated by directly transferring the captured data to the target model, and the yellow curve is the trajectory constructed by our method. Compare the edited result following on different constraint trajectory, we can see, that our result (d) is more similar to the original motion (b) than result (c) does.

4.2 Joint Limit Constraint

A joint value of a human is generally bounded to a valid domain, $[\theta_{\min}, \theta_{\max}]$. Once the limitation is broken, the motion will look unrealistic or uncomfortable. To solve this problem, we add joint limit constraint into our constraint generator. So the joint angle of the target motion is within the specified limits.

When the joint angle q exceeds the specified limit $[\theta_{\min}, \theta_{\max}]$, we reduce the q to the given limits. Thus, the corresponding constraint expression can be written as follows:

$$M(t) = \begin{cases} q & if\ q \in [\theta_{\min}, \theta_{\max}] \\ \theta_{\max} & if\ q > \theta_{\max} \\ \theta_{\min} & if\ q < \theta_{\min} \end{cases}$$

5. KALMAN FILTER RESOLVER

To obtain the reasonable and realistic motion, both the constraint generation and the constraint resolve are two equally important aspects. An efficient resolver need to adjust the pose according to the specified end-effector constraints in real time manner, while keep the characteristic of the original motion. For this purpose, we use the UKF [Wan97] (unscented kalman filter) to solve the constraint in real time.

UKF is a method of optimal state estimation, which can accurately handle the non-linear constraint, and minimize the difference of pose between before and after motion editing. In our method, the inputs are the captured data q_i^{org} and the specified constraint $M(t_i)$ at ith frame. Here, the captured data q_i^{org} is regard as state value and the constraint $M(t_i)$ as actual observation value. The output is the new desired pose. During the editing process, we can further control the final pose based on the input value of the process mode and the constraint expression [Liu05]. The algorithm can be described as follows:

Algorithm:

For each frame i,

1) Process model

The process model is defined as: $x_i^- = q_i^{org}$

Where x_i^- is the process model, equaling to the captured motion data at ith frame. Since the original motion contains excellent motion quality, we select the q_i^{org} as the process model. Thus we can easily estimate the similar target motion. In this step, if x_i^- is a n-dimension vector, we need to set a n×n diagonal matrix, P_i^-, as the input value. Here P_i^- relates to the displacement of each DOF in the editing process. Moreover, it can further control whether some DOF is changed before and after editing.

2) Calculation of the sample points

Given an n-dimension vector x_i^-, we can calculate m sample points χ_m, m = 2n +1. These sample points completely capture the mean and covariance of the vector x_i^-.

$$\chi_0 = x_i^-$$

$$\chi_m = x_i^- + (\sqrt{(n+\lambda)P_i^-})_m \quad m = 1, \cdots, n$$

$$\chi_m = x_i^- - (\sqrt{(n+\lambda)P_i^-})_{m-n} \quad m = n+1, \cdots, 2n$$

In which λ is a scaling parameter (see [Van01]).

3) Measurement model

Since the given constraints must be satisfied in the editing process, we define the constraint expression as the measurement model. That is,

$$Y_m = f(\chi_m) \quad m = 0, ..., 2n$$

Y_m is the predicted measurement value of the mth sample point. The mean y_i^- is approximated by a weighted sample mean:

$$y_i^- = \sum_{m=0}^{2n} W_m^m Y_m$$

The weight is given by:

$$W_0^m = \lambda/(\lambda + n)$$

$$W_0^c = \lambda/(\lambda + n) + 1 - \alpha^2 + \beta$$

$$W_m^m = W_m^c = 1/2(\lambda + n) \quad m = 1, ..., 2n$$

4) The gain of Kalman filter

$$P^{yy} = \sum_{m=0}^{2n} W_m^e (Y_m - y_i^-)(Y_m - y_i^-)^T$$

$$P^{xy} = \sum_{n=0}^{2n} W_m^c (\chi_m - x_i^-)(Y_m - y_i^-)^T$$

$$k_i = P^{xy}(P^{yy})^{-1}$$

5) Update of state value

Based on the specified constraint $M(t_i)$, new pose $\hat{x}_i$ at i-frame can be computed by:

$$\hat{x}_i = x_i^- + k_i(M_i^{des} - y_i^-)$$

6. SELF-COLLISION DETECTION

In this section, we present an efficient and real-time self-collision detection method. The method can be divided into two phase: the coarse phase and the fine phase. The coarse phase quick eliminates the meshes that are absolutely collision-free within a frame based on OBB. While the fine phase identifies how many meshes are indeed overlapping in all potential collision meshes.

6.1 Hierarchical Representation of OBB

6.1.1 Construction of Bounding Box

Before building the hierarchical representation of the human body, we need to decide what type of Bounding Volume (BV) to be used. Here, we apply a cost equation [Got96] to select a reasonable BV. For a human body, we believe that OBBs is a good choice. There are two reasons why we choose the OBB against other kinds of bounding volume:

(1) OBB can fit the original model as tightly as possible, thus the number of checking overlap in the fine phase will reduce significantly.

(2) The human model can be described as kinematic chain consisted of limbs, and every limb is nearly symmetrical, so the OBB hierarchy can be easily constructed and updated without any extra cost. Since the kinematic chain of human body is generally subdivided into limbs corresponding to the bones, and each limb includes a subset of triangle meshes relative to the whole triangle meshes of body, so we can construct an OBB for each limb and compute the size of OBB according to the subset of triangle meshes. An OBB is a rectangle box defined by three axes, three radiuses and a centre superposed to the centre of the corresponding limb. Because every limb is nearly symmetrical, we select the main axis of OBB along the connection orientation between the child joint and the parent joint, and then the other two axes can be obtained by the covariance matrix of the subset of the triangle mesh. As for the three radiuses, they can be computed by maximum and minimum extend of mesh subset along each axis. The construction of OBB is shown in figure 3. For the special limbs, such as the hip and shoulder, we note that they have two symmetrical and relative fixed bones, so the two bones can be replaced by a new bone to construct bounding box, as shown in figure 4.

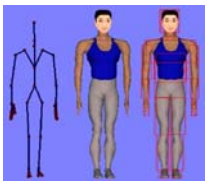


Figure 3. Skeleton, human body and OBB hierarchy model

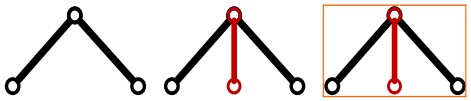


Figure 4. The OBB of hip: the black line means the bones of left hip and the right hip, respectively; the red line stands for the replacement bone; the yellow rectangle means the bounding box of the hip

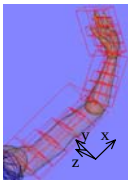


Figure 5. The OBB hierarchy of upper limb with five subdivisions along the bone, the coordinate is selected in terms of the upper arm, and x axis is the main axis

In order to eliminate substantially the collision-free meshes, we further subdivide each OBB into several parts along the main axis (the orientation of bone). Therefore the limbs is divided several parts by the sub boxes and these sub boxes can bound the limb more tightly than its parent box. Because we only need to test the pair of triangle meshes falling in the sub-box in the fine phase, the computational complexity is further lower. An example is shown in figure 5.

6.1.2 Obb Update

Whenever the change is applied to the freedom of human body, all bounding boxes that contain the affected joint need to be updated, so efficient update of OBB is crucial for the effectiveness of the collision detection. In the process of updating, the vertex position P_B of the bounding box and its sub-box are unchanged in the local coordinate system *B*, and the origin of the local coordinate system can be presented as a rotation matrix R_A^B and a translation vector T relative to the global coordinate system A. So for each update, the vertex position P_A in the global coordinate system can be computed by;

$$P_A = R_A^B \cdot P_B + T$$

So, our method can update the OBB at linear cost.

6.2 Collision Detection

In the coarse phase, we first detect collision between different OBBs, if the two OBBs have overlapped, we continue down the OBB hierarchy detecting collision between different sub-boxes, until all potential collision sub-boxes is detected. The phase can quick eliminate the collision meshes.

Collision detection between two OBB can be handled by the theorem of separating axis [Got96a]. For two OBB in space, there are 15 potential separating axes to be tested. Every OBB is projected onto these axes to form an interval, if the two intervals have not overlap, which means the two OBBs are collision-free.

For the human body containing N parts and N-1 joints, the number of OBB pairs that must be check is: $P = (N^2 - N)/2$. For example, in our model, N = 23, the number of limb pairs to be checked is 253. So the computation is huge. However, the human body is also different from the general chain, and the movements of its joints are all limited in a specified range. That means some pair of limbs are absolutely collision-free, for example the neck will never penetrate the thigh or the head. Based on this idea, we can utilize the limit of joint to pre-compute all potential collision-free limbs. Thus, the number of limb pairs which must be checked can be significantly reduced.

The above method works well in "rejection test", but if two objects have collision, the method needs to divide the OBBs continuously to detect potential collision. With the increasing of the number of sub-box, the performance of detection will slow down severely. For this reason, we use triangle-triangle detection [Tom97] to detect collision among triangles meshes in the fine detection phase.

7. Example

The articulated figure in our experience has a total of 23 limbs and each has 3 DOFs. The number of pairs should be checked is 253. Considering the kinematic reachability limitations, the number of remaining pairs will lower to 127. The computation reduces significantly.

In this experience we reuse the jump motion data to a target model. Figure 6 shows the original motion of the performer, and figure 7 shows the result that we transfer directly the captured data of straight jumping to the target model. Since the limb length of the virtual character is different from that of the performer, collision occurred: the hands penetrate his head. In figure 8, we drag the hand to a desired position, feed back the new constraint to constraint trajectory generator, and then build a new constraint trajectory

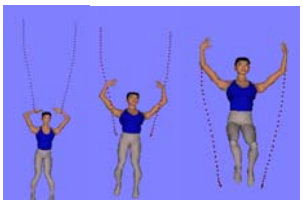


Figure 6. The original motion

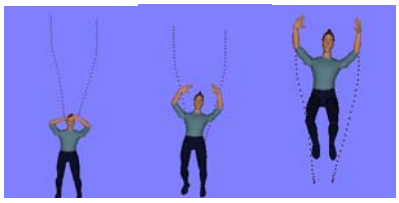


Figure 7 Transferring the original data to the target character, collision occurred

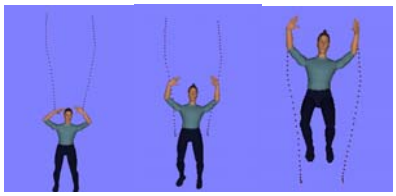


Figure 8: The final motion with collision free

8. CONCLUSION

In this paper, we provide a new method for editing the motion-captured data to produce the collision-free motion in real time. The central idea of our method is to integrate the issues of collision detection, constraint generation and constraint resolve to produce the collision-free motion. Further more, for each of these issues, we all provide an elegant solution. Comparing to other existing methods, our method not only can obtain the collision-free motion in real time, but also can preserve the original characteristics as many as possible and generate the new motion similar to the original motion.

9. ACKNOWLEDGEMENTS

This research is supported by National 973 project, Grant(2002CB312104); Key Cooperation Project of International Science and Technology (2005D-FA11060); Key Project of NSF (60533070); NSF of China, Grant (60573162, 60403042, 60473002); 863 Plan of China, Grant (22004AA115130, 2005AA-114010);National Special Item for Olympics, Grant (Z0004027040331,Z0004024040231); Beijing Natural Science Foundation (4051004, 4062032); and Knowledge Innovation Project 20056380 of Institute of Computing Technology, Chinese Academy of Sciences. We thank for the contributions to this work from all our colleagues in the Digital laboratory ICT of CAS.

10. REFERENCES

[Bad93] Badler, N.I, Hollick, M.J. and Granieri, J.P. Real-time control of a virtual human using minimal sensors. PRESENCE,:pp: 82-86, 1993.

[Brown01] Brown, J., Sorkin, S, Bruyns, C. Latombe, J., Montgomery, K., and Stephanides, M. Real-time simulation of deformable objects: Tools and application. In Comp. Animation, 2001.

[Bin98] Bindiganavale, R. and Badler, N.I.. Motion abstraction and mapping with spatial constraints. in CAPTECH'98 con.proc, pp: 70-82, 1998.

[Ban95] Bandi, S. and Thalmann, D. An adaptive spatial subdivision of the object space for fast collision of animated rigid bodies. in Eurographics'95, pp: 259-270, 1995.

[Can88] Canny, J.F. The complexity of robot motion planning. MIT Press, 1988.

[Cho00] Choi, K. and Ko, H. On-line motion retargeting. Journal of Visualization and Computer Animation 11, 5, pp: 223-235, 2000.

[Gle98] Gleicher, M. Retargetting motion to new haracter. Com.of ACM, pp. 33-42, 1998.

[Gle01] Gleicher, M. Comparing contraint-based motion editing methods. Graphical Models.63,2, pp. 107-134, 2001.

[Got96a] Gottschalk, S. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996

[Got96b] Gottschalk, S. Lin, M.C. and Manocha, D. OBBTree: A hierarchical structure for rapid interference detection. Conp. Graphics, pp: 171-180, 1996.

[Hua96] Huang, Z. Motion control for human animation. PhD thesis. EPFL-DI-LIG, 1996

[Liu05] Liu Li, Wang Zhao-qi, Zhu Deng-ming, Xia Shi-hong. An interactive motion retargeting method. in CASA'05 Conf.proc 2005.

[Lee99] Lee, J. and Shin, S.Y. A hierarchical approach to interactive motion editing for human-like figures. Com.of ACM, pp. 39-48, 1999.

[Mol97] Moller, T. A fast triangle-triangle intersection Test. Journal of graphics tools, pp: 25-30, 1997.

[Nat90] Naylor, B. Amanatides, J. and Thibault, W. "Merging bsp trees yield polyhedral modelling results", I Proc. of ACM Siggraph, , pp.: 115-241990.

[Neb00] Nebel, J-C. Realistic collision avoidance of upper limbs based on neuroscience models. Eurographics. 19(3). 2000.

[Qui94] Quinlan, S. Efficient distance computation between non-convex objects. in proceedings of international conference on robotics an automation. pp:3324-3329, 1994 .

[Sam89] Samet, H. Spatial data structures: quadtree, octrees and other Hierarchical Methods. Addison Wesley, 1989

[Shi01] Shin, H.J., Lee, J., Shin, S.Y., and Gleicher, M. Computer puppertry: an importance-based approach. ACM Transactions on Graphics 20, 2, pp. 67-94, 2001.

[Tak05] Tak, S. and Ko, H. A physically based motion retargeting filter. ACM Transactions on Graphics, Vol. 21, Issue 3, 2005

[Tom97]M.Tom. A fast triangle-triangle intersection test. Journal of Graphics Tools, 2(2):25-30, 1997

[Unu95] Munetoshi U. Ken A. and Ryozao T. Fourier principles for emotion-based human figure animation. conf.proc of ACM'95, pp: 91-96.1995

[Van97]Van, G. and Bergen, D. Efficient collision detection of complex deformable medels using AABB trees. J.of Graphics Tools, 2(4), pp: 1-13, 1997.

[Wan01] Wan, E.A. and Merwe, R.V.D. Kalman filter and neural netwroks. John Wiler&Sons. 2001.

[Zha94] .Zhao, J. and Badler, N.I. Interactive body awareness. Computer-aid design, 26(12), pp:861-867, 1994

Grafting Locomotive Motions

Shrinath Shanbhag
Indian Institute of Technology Bombay
India 400076
svs@it.iitb.ac.in

Sharat Chandran
Indian Institute of Technology Bombay
India 400076
sharat@cse.iitb.ac.in

ABSTRACT

The notion of transplanting limbs to enhance a motion capture database is appealing and has been recently introduced [Sha04], [Ike04]. A key difficulty in the process is identifying believable combinations. Not all transplantations are successful; we also need to identify appropriate frames in the different clips that are "cut-pasted." In this paper, we describe motion grafting, a method to synthesize new believable motion using existing motion captured data. In our deterministic scheme designed for locomotive actions, motion grafts increase the number of combinations by mixing independent kinematics chains with a base motion in a given clip.

Our scheme uses a cluster graph data structure to establish correlation among grafts so that the result is believable and synchronized.

Keywords

Motion capture, motion synthesis, cluster graph, motion grafting

1. INTRODUCTION

Movies and interactive applications such as games use virtual humanoid actors extensively. These virtual actors are modeled using hierarchical skeletons consisting of 30 or more degrees of freedom. Animating such characters is a daunting task at best. The animation data for such characters can be generated in one of three ways: (i) it can be produced procedurally by simulating various physical processes, (ii) it can be handcrafted painstakingly by skilled animators using forward/inverse kinematics based systems or (iii) it can be acquired directly from a live performer using motion capture devices. Method (iii) has gained wide acceptance in recent times because motion capture is the fastest way to generate rich, realistic animation data. In addition motion capture (mocap) techniques can capture even individual nuances of a performer and thus produce very realistic animation.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

Having said, this, we recognize that human beings are active entities that produce innumerable actions. A characteristic of humans is that we perform logically distinct and kinematically unrelated actions in parallel. For example, consider "a hand wave sequence." A performer can wave his hands when in different postures — while standing, sitting, talking, and walking. The actions in such cases may be viewed as compositional. The number of such compositional actions tends to grow exponentially. Additionally each action can be performed in multiple styles. For example the hand wave may be performed at different rates. Variations of the same basic motion often depend upon internal factors such as moods and external factors such interaction contexts or physical constraints.

Traditionally, however, data for each actor action is acquired separately. In contrast to the discussion in the previous paragraph, such a scheme has the disadvantage that virtual actor actions are limited to the number of sequences originally captured. This drawback directly impacts interactive applications. In such applications only a few motion sequences corresponding to key actions performed by the virtual actors are used. The reason for this is pragmatic. It is difficult to anticipate and act out "all" the different combinations of actions that would ever be required during runtime. Capturing all the possible correlated combinations is often neither possible nor desired despite the obvious

advantages of capturing a large number of combinations.

This Paper

The obvious way to deal with motion composition is to cut and paste motion segments across existing mocap clips. However, such an attempt is likely to result in motions that do not look human as a result of inherent lack of cross-body correlation.

The correlation in a real locomotive sequence may be due to active intentions on part of the actor or as the result of passive reflex. Examples of intentional correlation occur in movements such as relaxed walking, where arms swing out of phase with the legs for energetic reasons. This is a gait that is chosen by the actor, and can be broken at will – for example, to scratch or to reach. Reflex correlations occur as a result of the body reacting to maintain equilibrium. For example, the arms may be extended out in order to balance a fall. If the arm movement in this case is replaced with some other arm movement, the resulting motion may not look human. In either case the correlations play an important role in defining believability of the final motion.

While researchers in the field of behavioral animation have built systems, [Blu95][Per96], that take advantage of parallelism in actions, attempts at automatic composition are recent [Sha04][Ike04]. In this paper we describe our method to synthesize new motions by composing together different actions onto a base clips, taking into consideration the problems discussed above. Our solution is based on a scheme that breaks down the original problem into manageable parts as shown in Figure 1. The net result is (see video) that it generates believable grafts.

To our knowledge motion grafting was first discussed in our own prior unpublished work [Sha04]. An interesting implementation has been subsequently described in [Ike04]. The work presented here complements the work in [Ike04] in the following ways:

- For increased quality, we target only motions that have running, jumping, and walking motions. Several unsuccessful transplants are reported in [Ike04].
- The randomization rules to generate new motion are not used in our work. Instead currently we have used a Cartesian product of upper and lower body classification to generate candidate grafts.
- Instead of using an SVM based classification to determine successful transplants, we use the intrinsic correlation available in cluster graphs.

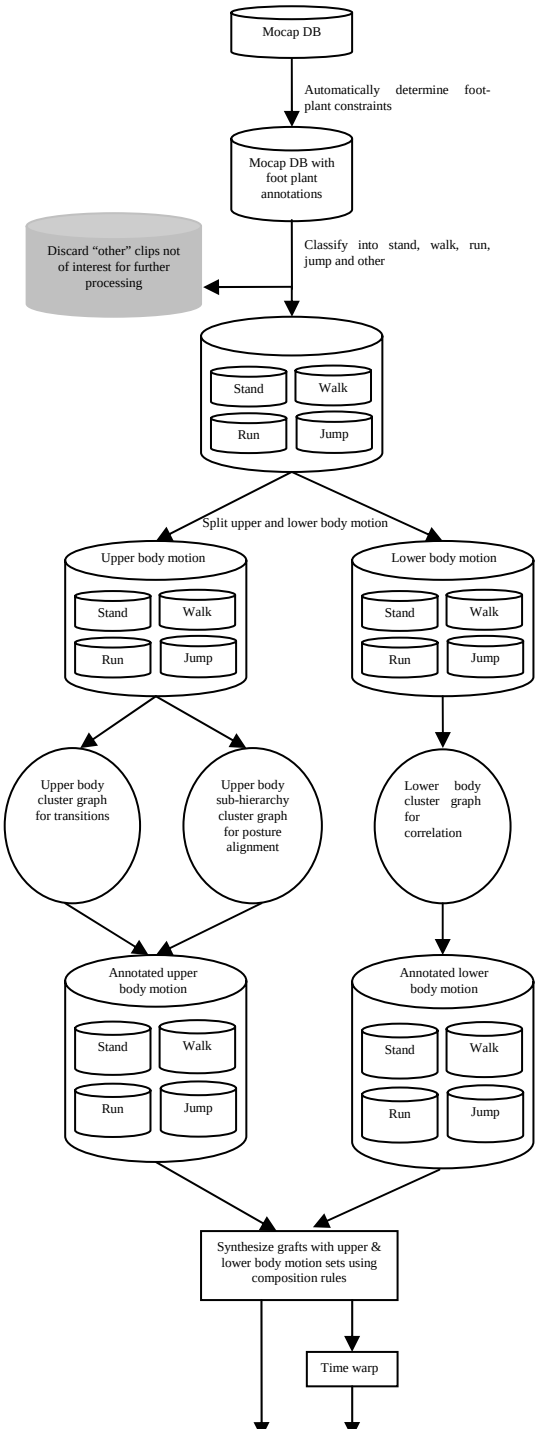


Figure 1 Grafting framework. Our scheme starts with a discovery of locomotive motions from the motion capture database. After classifying independent kinematics chains, a cluster graph data structure is used to correlate seemingly different motions. Correlation is key to generating believable grafts. An optional time warp enables "scaling" in time.

The rest of this paper is organized as follows. We first mention the related work in this area. Next we briefly describe our cluster graph [Bas05] data structure. An overview of our process follows in Section 4. Details of the steps such as determination of independent kinematics chains are described in Section 5. Sample results follow to demonstrate the efficacy of our method (results are best viewed in the accompanying videos).

2. Background

The history of research in humanoid animation dates back more than 15 years. Motion synthesis methods can be broadly classified as kinematics based, dynamics based and constraint based methods. In addition there are also hybrid methods which mix one or more of the other techniques. More recently, methods based on motion captured sequences have been developed.

Behavioral Animation

Behavioral animation researchers have observed and described ways to synthesize parallel actions. [Blu95] describes the use of lock variables to share degrees of freedom between motor skills. [Per96] explicitly allow for action compositing in Improv. Therein they describe a layered architecture for enabling parallel execution of independent actions. The primary difference between these works and ours lies in the fact that they deal with motion sequences explicitly designed to work independently. Defining high fidelity motion sequences is non-trivial. [Per96] construct motion sequences using combinations of sine, cosine and coherent noise. Noise is used to generate variations in motion. Sequences generated in this manner, though not repetitive, do not contain individualistic nuances of the performer. We choose to work with motion captured data which is richer. However the independent actions are not well defined here.

Gait Synthesis Techniques

[Mul02] provide an excellent survey of computer animation of human walking. [Sun01] describe a low-level gait generator based on sagittal elevation angles, which allows curved locomotion to be created easily. They also describe an inverse motion mapping algorithm that allows motion to be adapted to uneven terrains. In addition they describe a higher level control frame work that allows motion requirements to be specified at a high level by sketching the desired path. [Hod95] describe an algorithm that allows simulation of running, bicycling and vaulting. The simulation is achieved through control algorithms that cause physically realistic models to perform the desired behavior. [Fal01] describe a method to combine various physically based simulation controllers into a unified framework.

Mocap based Motion Editing Techniques

Mocap based techniques, unlike synthesis techniques described above, start with an existing motion and adapt it to different requirements. Researchers working in this field have proposed a number of innovative techniques adapting signal processing methods or employing constraint based solvers. [Bru95] have successfully applied techniques from image and signal processing domains to designing, modifying and adapting animated motions. [Unu95] describe a method for modeling human figure locomotion's with emotions. Herein Fourier expansions of experimental data of actual human behaviors serve as a basis from which to interpolate or extrapolate the human locomotion's. [Wit95] describe a simple technique for editing captured animation based on warping of the motion parameter curves. [Gle97][Gle98] use space-time constraint formulations to modify captured motion and retargeting motion to new characters with different segment lengths. [Gle01] provides a comparison of constraint based motion editing methods. [Lee99] describe a hierarchical framework for adapting existing motion of humanoids to externally specified constraints. [Shi03] describe a method that takes into consideration physical principles to touch up synthetically generated motion, so as to make it more plausible.

Mocap Based Synthesis Techniques

Motion synthesis techniques create new motion from existing motion data. [Kov02a], [Lee02], [Ari02] describe techniques to create new motion sequences from a corpus of motion data. Each technique essentially clusters similar motion into nodes. The next phase builds a graph of nodes, where each edge represents a transition between nodes. A walk through the cluster node graph results in synthesis of new motion sequences. The techniques differ in metrics used for clustering, pruning schemes and control criteria for node walk. [Pul00][Pul02] describe a scheme for synthesizing missing degrees of freedom and adding details to specified degrees of freedom, for a roughly specified motion sequence. Their method uses the various correlation between the various degrees of freedom (DOF's) within each motion sequences. Forsyth et. al [2] describe a technique using a novel search method based around dynamic programming to interactively synthesize motion from annotations. Here the system synthesizes motion corresponding to an annotated timeline painted by the user.

[Gle03] present a technique that preprocesses a corpus of motion capture examples into a set of short clips that can be concatenated to make continuous streams of motion. The resulting simple graph structure can be used in virtual environments where control and responsiveness are more important than accuracy. [Bra00] use machine learning techniques to model motion sequences as stylistic HMM parameterized by a style vector. Motion can be synthesized in a number of ways - a random walk over the HMM states, by trying to match given input sequence to an optimum set of HMM states etc. However, as the method is statistical there is no direct control over the desired motion.

3. CLUSTER GRAPH

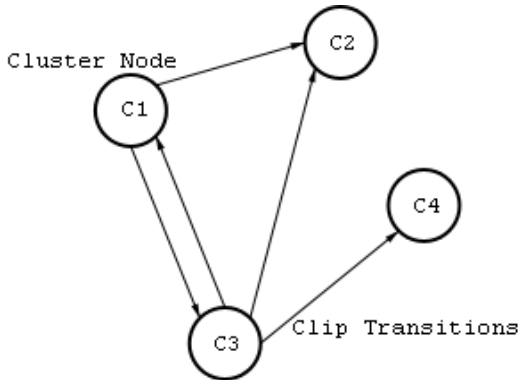


Figure 2 The cluster graph data structure

We describe the cluster graph [Bas05] in brief as our grafting technique uses cluster graphs extensively. The cluster graph is a versatile data structure that we use to cluster together frames from different clips based on similarity. A cluster graph

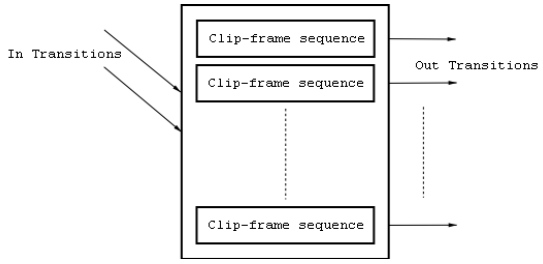


Figure 3 A cluster graph node

automatically chops individual clips and collects similar sub-clip sequences together. Each subsequence could be even half a dozen frames long. Seemingly unrelated clips are brought together into these nodes increasing the choices available for re-synthesis.

Nodes in a cluster graph contain frames from one or more clips. Frames within a node are "similar," that is, the error between any two frames is below a threshold. Edges are obtained from the natural sequential ordering of clips within nodes. Figure 2 shows an example.

Within a node (Figure 3), the frames are sorted by clips and time. Contiguous sequences of frames are collected together into a structure called clip-frame sequence. We maintain out-transitions for each clip-frame sequence for each cluster node. Maintaining one transition per clip-frame sequence automatically prunes away transitions from contiguous frames.

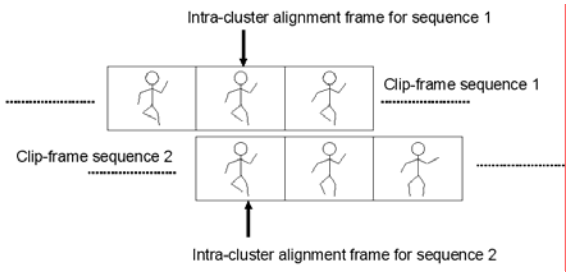


Figure 4 Intra-cluster alignment frames

Once clip-frame sequences are clustered in a graph, each cluster node contains similar frames. We then find an intra-cluster alignment frame, (Figure 4) for each clip in the cluster, using a correlation procedure. This alignment frame, is the best point of transition between any two clip-frame sequences in the cluster.

Cluster graph advantages

Cluster graph nodes contain clip-frame sequences from different mocap sequences clustered together based on similarity. These clustered clip-frame sequences establish correlation amongst clips.

Cluster graphs provide a level of granularity smaller than those of motion graphs [14]. Traditional motion graphs record only one transition point between each pair of clips. As a result this data structure has been used to find transitions between two clips without enforcing a hard time constraint. For the real-time version, time is an important factor. We may need to transition between two clips at precise, or more controlled instants in time. Therefore it is useful to maintain as many distinct transition points as possible. Note further that transition points occur in bunches. Cluster graphs prune multiple transition points lying very close to each other temporally.

4. METHOD OVERVIEW

Our aim is to allow motion re-synthesis by selectively grafting motion signals captured on

different actions on to a base signal. In order to do this we use a collection of techniques and steps as illustrated by Figure 1.

1. We start with a database of motion captured clips.
2. Every clip in the mocap database is preprocessed to detect foot plant constraints. We annotate the clips with this information.
3. We classify the clips based on the foot plant annotation into the following categories – "stand", "walk", "run", "jump" and "others." The clips of interest to us are the ones labeled "stand", "walk", "run" or "jump". We discard the clips labeled "other".
4. We separate the upper body and lower body motion signals based on their respective *independent kinematics chains*. We retain the respective upper and lower body pair correspondence during this process for later use.
5. We create a cluster graph for lower body motion. We use this to obtain correlation information amongst lower body motion. Lower body motion forms the base clip upon which grafting occurs.
6. We create two cluster graphs for the upper body motion sets. The first is created with the entire upper body and is used for grafting transition information. However, only a subset of transitions will not violate self penetration. A second cluster graph is created with just the *root – lowerback – upperback – torso* joint chain, to conservatively estimate safe grafts.
7. We synthesize grafts as a Cartesian product of upper body and lower body motion sets. We use the composition rules explained below.
8. As an optional last step we allow the animator to accept or reject generated clips before enhancing the motion database.

4.1.1 Composition Rules

Our categorization of motion results in upper and lower body motion divided into four sets corresponding to "stand", "walk", "run" and "jump". We synthesize graft motion grafts by taking certain conservative Cartesian products of upper and lower body motion sets:

1. (Upper body motion set "stand") X (Lower body action sets "stand", "walk", "run" and "jump").
2. (Upper body motion set "walk") X (Lower body motion sets "walk" and "run")
3. (Upper body motion set "run") X (Lower body motion sets "walk" and "run")
4. (Upper body motion set "jump") X (Lower body motion set "jump")

5. GRAFTING

Grafting is the process of synthesizing motion. It essentially involves masking out the original base clip signal for the selected set of DOF's and replacing them with those from a different clip. In this section we give some details of the modules in section 4.

Independent Kinematic Chains

We observe that a skeletal hierarchy essentially contains several kinematics chains as in Figure 5. For example nodes [root, lhipjoint, lfemur, ltibia, lfoot, ltoes] constitutes a kinematics chain. For different chains having a common root, the kinematics inter-dependency is restricted to the

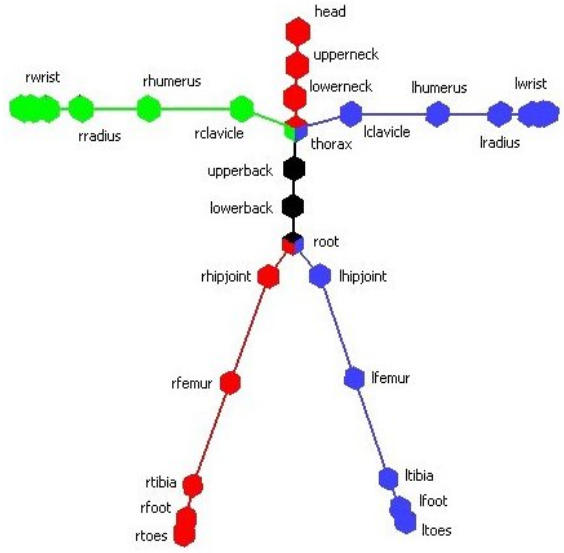


Figure 5 Hierarchical skeleton with independent kinematics chains demarcated

common roots DOF variables. The chains are unaffected by DOF variables at non root nodes. We call such chains "independent chains." The motion signals captured for each independent chain constitutes an "independent action." We think of a motion captured sequence to be composed of several "independent actions" running in parallel. We graft motion on to DOF's of these independent kinematics chains. Cross body correlation creates

inter-dependence amongst independent actions. We use independent kinematics chains to maintain correlation. We also use these kinematics chains to create sub-hierarchy cluster graphs.

Six such kinematics chains are defined in Figure 5 (seen best in color).

Chain 1: Thorax … Head

Chain 2: Thorax … LWrist

Chain 3: Thorax … RWrist

Chain 4: Root … Thorax

Chain 5: Root … LToes

Chain 6: Root … RToes

Clip classification

We use the lower body kinematics chains for clip classification. One of the important characteristics of lower body motion is the foot-plant constraint. We use foot-plant constraint pattern matching to classify clips. We make the following observation regarding foot-plant constraints.

1. Standing Stationary: Both feet remain planted.
2. Walking: Alternate feet are planted passing through a double step pose.
3. Running: Alternate feet are planted with intervening stages of both feet being of the ground.
4. Jumping: Both feet are either planted or in air simultaneously.

We encode the foot-plant behavior as a set of string symbols and use simple string matching to classify clips.

5.1.1 Detecting foot plant constraints

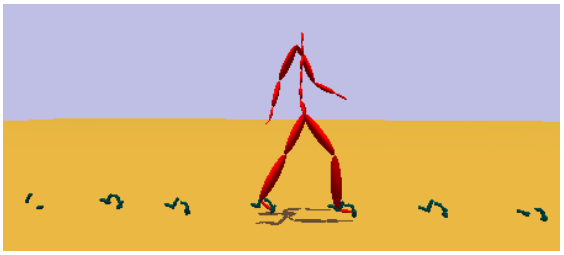


Figure 6 Frames with heel or ball joints stationary detected as foot-plant frames

We identify foot plant constraints by first identifying frames with zero crossings for vertical displacement (the y axis in our case). We select all frames which are close to the ground, within a given threshold. This forms the seed set of foot plant frames. For most normal walk sequences, we observe that the foot is placed on the ground for more than one frame. However, in a motion captured sequence, the foot positions may not coincide exactly due to foot skate. [Kov02b] describe a technique to identify and correct foot skate. We use a simpler method. From the initial set of foot plant frames obtained above, we sequentially search in both directions and cluster frames, near the seed foot plant frame, where the magnitude of the displacement vector is below a given threshold value. We stop the search at the first frame that fails the test. We then cluster together, like foot plant frames based on their sequence in the clip.

6. RESULTS

Our motion capture database, after categorization, consists of more than a 100 clips from the CMU motion capture database. As noted earlier the cluster graph data structure chops individual clips into smaller clip-frame sequences each of which can be as small as half a dozen frames. This combined with the ability of the cluster graph to detect and encode loops, results in an order or two magnitude increase

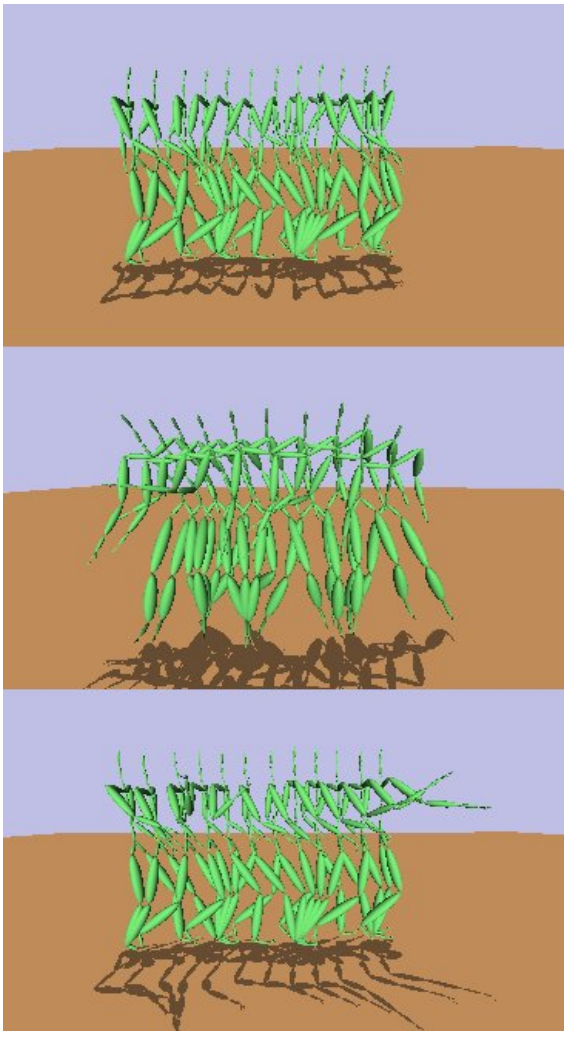


Figure 7 The walk clip (top) is used as the base clip onto which the basket ball dribble (middle) is grafted to synthesize new (bottom) clip.

in the large number of paths through the structure.

Figure 7 and Figure 8 are representative of the results obtained using our method. In Figure 7 The walk clip (top) is used as the base clip onto which the basket ball dribble (middle) is grafted to synthesize new (bottom) clip. In Figure 8 The normal walk clip (top) is used as the base clip onto which arm motions from the exaggerated stride (middle) is grafted to yield the marching like motion (bottom). As can be seen (from the accompanying videos) the results are fairly believable and smooth.

7. CONCLUSION

In this paper we have described our method to enhance a collection of motion captured clips by synthesizing new motion. Our synthesis technique composes together motion onto independent kinematics chains of a base clips. Our composition rules allow us to take into account the correlation between different actions. We use cluster graphs extensively to obtain correlation information.

8. REFERENCES

[Ari02] Arikan, O. Arikan and Forsyth, D. A. Interactive Motion Generation from Examples. In Proc. of Siggraph '02, pp. 483 - 490, 2002.

[Ari03] Arikan, O., Forsyth, D. A., and O'Brien, J. F. Motion Synthesis from Annotations. In Proc. of Siggraph '03, pp. 483 - 490, 2003.

[Bas05] Basu S. K., Shanbhag, S., and Chandran S., Search and Transitioning for Motion Catpured Sequences. ACM Symposium on Virtual Reality Software and Technology 2005.

[Blu95] Blumberg, B.M and Galyean, T. A., Multilevel Direction of Autonomous Creatures for Real-Time Virtual Environments, Proceedings of Siggraph '95, 1995, pp. 47-54.

[Bra00] Brand, M. and Hertzmann, A. Style machines. In Proceedings of Siggraph '00, 2000.

[Bru95] Bruderlin, A. and Williams, L. Motion Signal Processing. In Proc. of Siggraph '95, 1995.

[Fal01] Faloustos, P. , van de Panne, M., and Terzopoulos, D. Composable controllers for physics-based character animation. In Proceedings of Siggraph '01, August 2001.

[Gle97]Gleicher, M. Motion Editing with Spacetime Constraints.In Proc. of the 1997 Symposium on Interactive 3D Graphics.

[Gle98] Gleicher, M. Retargetting Motion to New Characters. In Proc. of Siggraph '98, 1998.

[Gle01] Gleicher, M. Comparing Constraint-based Motion Editing Methods. Graphical Model, 63(2):107 - 134, 2001.

[Gle03] Gleicher, M, Shin, H. J., Kovar, L., and Jespen, A. Snap together motion: Assembling run-time animation. In 2003 Symposium on Interactive 3D Graphics, April 2003.

[Hod95] Hodgins, J. K., Wooten, W. L., Brogan, D. C. , and O'Brien, J. F. Animating human athletics. In Proceedings of Siggraph '95, 1995.

[Ike04] Ikemoto, L., and Forsyth, D., A., Enriching a Motion Collection by Transplanting Limbs, Proc. ACM Symposium on Computer Animation, 2004.

[Inm89] Inman, V. T. , Ralston, H. , and Todd, F. Human Walking. Lippincott Williams & Wilkins, 1989.

[Kov02a] Kovar, L., Gleicher, M., and Pighin, F. Motion Grahs. In Proc. of Siggraph '02, 2002.

[Kov02b] Kovar, L., Schreiner, J., and Gleicher, M. Footskate cleanup for motion capture editing. In Proceedings of the 2002 ACM Symposium on Computer Animation (SCA), July 2002.

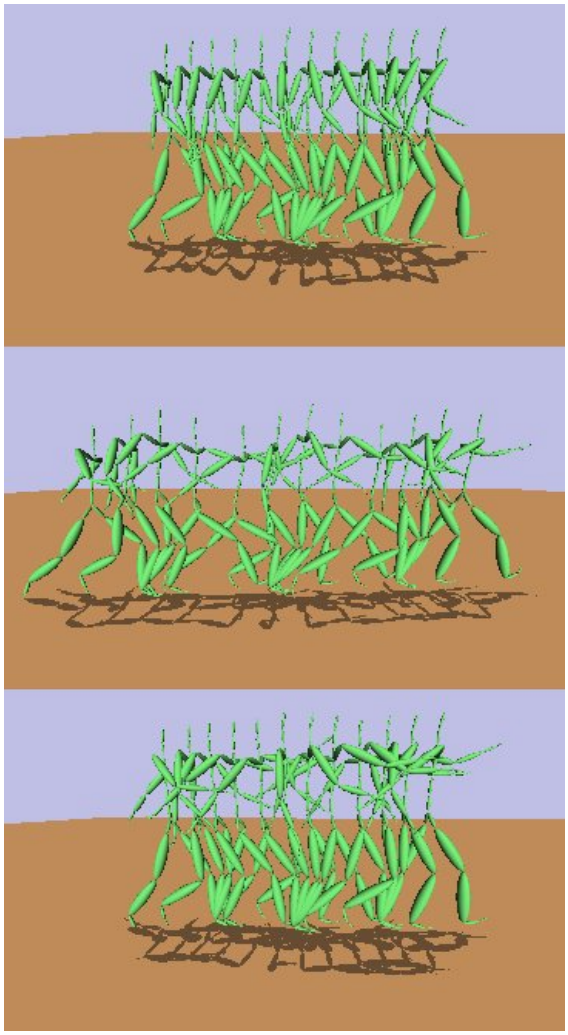


Figure 8 The normal walk clip (top) is used as the base clip onto which arm motions from the exaggerated stride (middle) is grafted to yield the marching like motion (bottom)

[Lee99] Lee, J. and Shin, S. Y. A Hierarchial Approach to Interactive Motion Editing for Human like Figures. In Proc. of Siggraph '99, 1999.

[Lee02] Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K., and Pollard, N. S. Interactive Control of Avatars Animated with Human Motion Data. In Proc. of Siggraph '02, 2002.

[Mul02] Multon, F., France, L., Cani-Gascuel, M.-P., and Debunne, G. Computer animaton of human walking: A survey. Technical Report 3441, INRIA, June 1988.

[Per96] Perlin, K and Goldberg, A., Improv: A System for scripting interactive actors in virtual world, Proceedings of Siggraph '96, 1996, pp. 205-216.

[Pul00] Pullen, K. and Bregler, C. Animating by Multi-level Sampling. In Proc. of IEEE Computer Animation 2000, 2000.

[Pul02] Pullen, K. and Bregler, C. Motion Capture Assisted Animation: Texturing and Synthesis. In Proc. of Siggraph '02, 2002.

[Shi03] Shin, H. J., Kovar, L., and Gleicher, M. Physical touch-up of human motion. In Pacific Graphics 2003, October 2003.

[Sha04] Shanbhag, S., and Chandran, S., Parallel Action Synthesis for Skeletally Animated Characters, Technical Report IIT Bombay, February 2004.

[Sun01] Sun, H. C. and Metaxas, D. N. Automating gait generation. In Proceedings of Siggraph '01, August 2001.

[Unu95] Unuma M., Anjyo, K., and Takeuchi, R. Fourier principles for emotion based human figure animation. In Proceedings of Siggraph '95, 1995.

[Wit95] Witkin, A. and Popovic, Z. Motion Warping. In Proc. Of Siggraph '95, 1995.

Reverse Catmull-Clark Subdivision

Sandrine Lanquetin
LE2I, UMR CNRS 5158
Université de Bourgogne
BP 47870
21078, DIJON, France
sandrine.lanquetin@u-bourgogne.fr

Marc Neveu
LE2I, UMR CNRS 5158
Université de Bourgogne
BP 47870
21078, DIJON, France
marc.neveu@u-bourgogne.fr

ABSTRACT

Reverse subdivision consists in constructing a coarse mesh of a model from a finer mesh of this same model. In this paper, we give formulas for reverse Catmull-Clark subdivision. These formulas allow the constructing of a coarse mesh for almost all meshes. The condition for being able to apply these formulas is that the mesh to be reversed must be generated by the subdivision of a coarse mesh. Except for this condition, the mesh can be arbitrary. Vertices can be regular or extraordinary and the mesh itself can be arbitrary (triangular, quadrilateral…).

Keywords

Reverse Subdivision, multiresolution, Catmull-Clark scheme.

1. INTRODUCTION

Subdivision surfaces were introduced in 1978 by Catmull-Clark [Cat78] and Doo-Sabin [Doo78] as an extension of the Chaikin algorithm [Cha74]. These surfaces are widely used in character animation (such as Geri's Game © or Finding Nemo ©) to smooth out models. Indeed, from a coarse mesh, successive refinements give finer meshes. A sequence of subdivided meshes converges towards a smooth surface called limit surface. Since the earliest subdivision surfaces in 1978, many subdivision schemes were proposed. Some are approximating and others are interpolating (i.e. control vertices of successive meshes belong to the limit surface).

The main advantage of the Catmull-Clark scheme over the triangle based subdivision such as the Loop scheme [Loo87] is that the control mesh faces can have an arbitrary number of edges. This is an important feature in modeling because most of the time designers build their model by symmetry as shown in Figure 1. Moreover subdivision surfaces are more and more used in CAGD and in this field; most meshes are quadrilateral, in coherence with parametric surfaces (Bezier, B-splines, NURBS…).

Subdivision methods produce an increasingly fine sequence of meshes. On the contrary, it can be interesting to pass quickly from a mesh to a coarser one according to the point of view for example. Using a local formula for decreasing the resolution of a mesh is a crucial element for the implementation of multiresolution surfaces. It reverses the subdivision process. While formulas for subdividing meshes are local, the existence of local formulas for the respective reverse subdivision is less evident.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

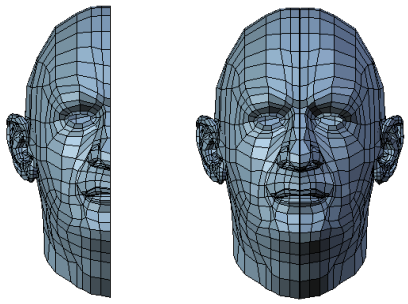


Figure 1. A character head build by symmetry.

There are several global methods such as multiresolution methods [Lou97], as Loop reverse subdivision [Mon04]. The interest for the local methods however is new. Samavati and Bartels

determined the local reverse subdivision masks for the Butterfly and Loop scheme restricted to regular vertices (valence 6) in [Sam02]. Samavati et al. focused on the Doo-Sabin scheme for arbitrary meshes in [Sam02]. Samavati, Pakdel, Smith and Prusinkiewicz [Sam03] propose a local method for the Loop scheme which consists in locally reversing the formula for a given set of vertices.

In this paper, we focus on Catmull-Clark and develop a local method to reverse this subdivision. Section 2 overviews the Catmull-Clark scheme. Then, we present our method for reversing Catmull-Clark scheme in Section 3. Thus obtained results are shown in Section 4, illustrating different cases that can occur.

2. BACKGROUND

The Catmull-Clark subdivision scheme

The Catmull-Clark subdivision scheme generalizes the generation algorithm of cubic B-splines to surfaces; it is based on the tensor product bicubic spline [Dyn90]. The rules of Catmull-Clark were first defined for meshes with quadrilateral faces but they can easily be generalized to arbitrary polygonal meshes. The *valence* of a vertex is the number of vertices connected to this vertex by an edge. In the case of Catmull-Clark, if the vertex valence is not four the vertex is denoted as an *extraordinary vertex*. Even if the initial mesh is not quadrilateral, meshes generated at each subdivision level are quadrilateral.

Let M^k be the mesh at the subdivision level k. Each vertex of M^{k+1} can be associated to a face, an edge or a vertex of M^k. These vertices are respectively called face point, edge point or vertex point.

- A face point denoted f_j^{k+1} is computed as the average of the vertices of this face:

$$f_j^{k+1} = \frac{1}{|F_j|} \sum_{f_i^k \in F_j} f_i^k \qquad \textbf{(1)}$$

- An edge point e_j^{k+1} is computed as the average of the endpoints of this edge and the face points of the two incident faces of this edge :

$$e_j^{k+1} = \frac{v^k + e_j^k + f_j^{k+1} + f_{j-1}^{k+1}}{4} \qquad \textbf{(2)}$$

where v^k and e_j^k are the endpoints of the edge on the k^{th} subdivision level and f_{j-1}^{k+1} and f_j^{k+1} are the newly computed face points of the faces incident to this edge.

- A vertex point is a weighted average of its incident vertices of the same level and of the face points of the incident faces

$$v^{k+1} = \frac{n-2}{n} v^k + \frac{1}{n^2} \sum_{j=0}^{n-1} e_j^k + \frac{1}{n^2} \sum_{j=0}^{n-1} f_j^{k+1} \qquad \textbf{(3)}$$

where f_j^{k+1} are the newly computed face points, e_j^k are the neighbours of v^k on the same subdivision level and n is the valence of the vertex v^k.

Figure 2 illustrates these notations for a vertex v^k with valence n.

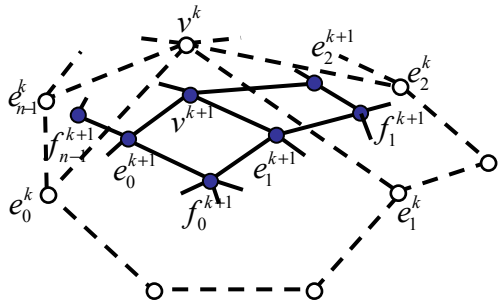


Figure 2. Catmull-Clark Subdivision for a vertex v^k with valence of n.

The valence of a vertex point remains the same after subdivision i.e. $\#v^k = \#v^{k+1}$ and $\#v_j^k = \#v_j^{k+1}$ where $\#$ denotes the valence of a point. The valence of edge points are always four and the valence of face points corresponds to the number of edges of the face. Figure 3 shows the successive meshes obtained with the Catmull-Clark subdivision.

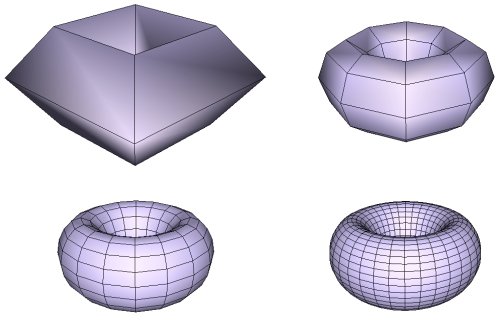


Figure 3. Catmull-Clark subdivision applied on a torus mesh.

Boundaries

The formulas used for vertices on a mesh boundary are different from those for interior vertices. The notations used are the same than previously but this time v^k, e_0^k and e_1^k form a boundary of the mesh (Figure 4).

The formula used for computed boundary vertices with Catmull-Clark scheme are those of cubic B-spline curves:

$$v^{k+1} = \frac{1}{8}e_0^k + \frac{3}{4}v^k + \frac{1}{8}e_1^k$$
$$e_0^{k+1} = \frac{1}{2}v^k + \frac{1}{2}e_0^k$$
$$e_1^{k+1} = \frac{1}{2}v^k + \frac{1}{2}e_1^k$$

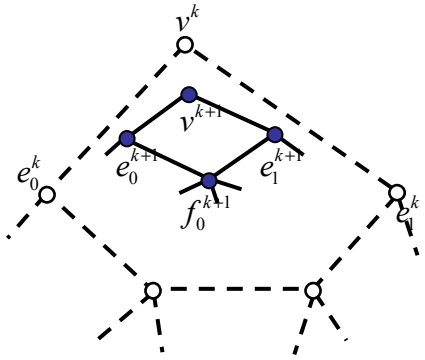


Figure 4. v^k, e_0^k and e_1^k form a boundary of the mesh.

3. Reverse formula for Catmull-Clark subdivision

Method

For the reverse subdivision process, it is necessary to expand a formula in which v^k is only determined from v^{k+1} and its neighbourhood e_j^{k+1}, f_j^{k+1} $j \in [\![0, n-1]\!]$ with n the valence of v^{k+1}. Let v^k be a vertex with valence n and M^k be an arbitrary mesh as shown in Figure 5.

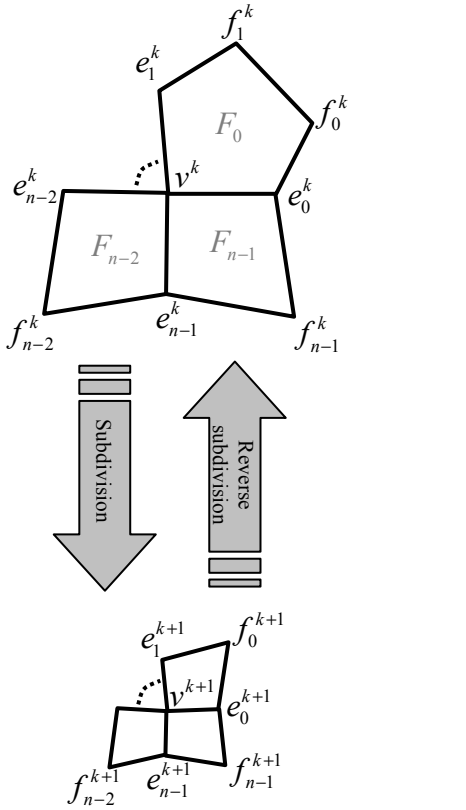


Figure 5. General neighbourhood for an extraordinary vertex.

Ordinary and extraordinary vertices

In the reverse problem, we know vertices of the level $k+1$ and we want to find the vertex v^k using a formula which respects the following conditions:

- The formula is an affine operation
- The neighbours e_j^{k+1} of v^{k+1} in the formula must have the same weight as in Equation **(3)** and the centroid f_j^{k+1} of faces F_i incident to v^{k+1} in the formula must also have the same weight
- The application of the reverse formula must exactly reconstruct v^k i.e. the subdivision of the vertex v^k generates the vertex v^{k+1} and the reverse subdivision of the vertex v^{k+1} gives v^k.

The second condition yields to the diagram shown in Figure 6. This diagram is called mask and shows the coefficient to apply on vertices.

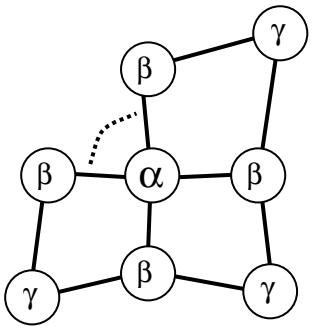


Figure 6. Reverse Mask corresponding to the general neighbourhood of Figure 5.

Let α, β, γ be respectively the coefficients for the vertex point, edge points and face points. For the third condition presented above, the following equation must hold:

$$\alpha v^{k+1} + \beta \sum_{j=0}^{n-1} e_j^{k+1} + \gamma \sum_{j=0}^{n-1} f_j^{k+1} = v^k .$$

From Equation **(1)** to **(3)**, we obtain:

$$\left(\frac{(n-2)\alpha}{n} + \frac{n\beta}{4} \right) v^k + \left(\frac{\alpha}{n^2} + \frac{\beta}{4} \right) \sum_{j=0}^{n-1} e_j^k + \left(\frac{\alpha}{n^2} + \frac{\beta}{2} + \gamma \right) \sum_{j=0}^{n-1} f_j^{k+1} = v^k$$

.

This equation yields to the following system:

$$\begin{cases} \frac{n-2}{n}\alpha + \frac{n}{4}\beta = 1 \\ \frac{1}{n^2}\alpha + \frac{1}{4}\beta = 0 \\ \frac{1}{n^2}\alpha + \frac{1}{2}\beta + \gamma = 0 \end{cases}$$

This can be written with matrix:

$$A \times \Theta = B \quad \text{where} \quad A = \begin{bmatrix} \frac{n-2}{n} & \frac{n}{4} & 0 \\ \frac{1}{n^2} & \frac{1}{4} & 0 \\ \frac{1}{n^2} & \frac{1}{2} & 1 \end{bmatrix}, \quad \Theta = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

and $B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

The determinant of the matrix A is $\det(A) = \frac{n-3}{4n}$.

So the system can be solved only when $n \neq 3$.

For $n \neq 3$, we find:

$$(\alpha, \beta, \gamma) = \left(\frac{n}{n-3}, \frac{-4}{n(n-3)}, \frac{1}{n(n-3)} \right)$$

Finally, for an ordinary or extraordinary vertex with valence $n \neq 3$:

$$v^k = \frac{n}{n-3} v^{k+1} + \frac{-4}{n(n-3)} \sum_{j=0}^{n-1} e_j^{k+1} + \frac{1}{n(n-3)} \sum_{j=0}^{n-1} f_j^{k+1} \qquad \textbf{(4)}$$

This allows to compute v^k in most of the cases.

Vertices with valence $n = 3$

If $n = 3$, the system of the previous section cannot be reversed because the determinant is equal to zero. So, we have to find another way to construct v^k. The solution we chose is to perform the rebuilding of the mesh at the level k in two stages:

- First, vertices of valence $n \neq 3$ are constructed
- Then, vertices with valence $n = 3$ are treated

Let v^k be a vertex with valence $n = 3$. Figure 7 illustrates notations used in this section. The mesh at subdivision level k is drawn with dotted lines and the subdivided mesh is drawn with solid lines.

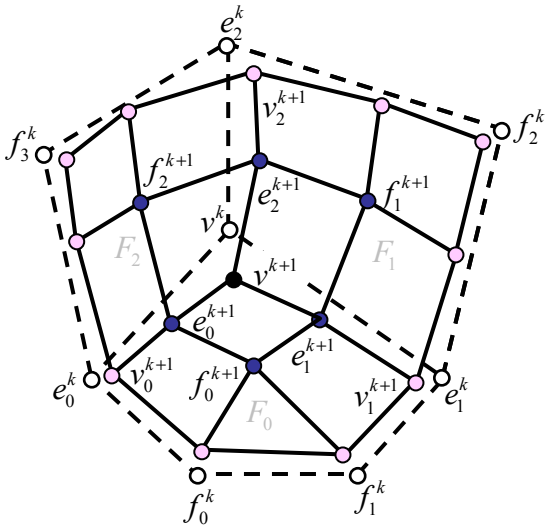


Figure 7. Notations used in the case $n = 3$.

We now will explain the method used when $n=3$. The vertex v_j^{k+1} associated to e_j^k, $j \in [\![0,2]\!]$ at the level $k+1$ necessarily has the same valence. Thus even if e_j^k is not already reconstructed, we know its valence. Moreover, from the previous section, we know that only vertices with valence $n=3$ are not reconstructed. This property is used to classify the vertices in two categories:

- Either there is at least one vertex among the incident vertices e_j^k of the vertex v^k, $j \in [\![0,2]\!]$ which is already constructed (with valence $n=3$ or not)
- Or there is no vertex among the incident vertices e_j^k of the vertex v^k, $j \in [\![0,2]\!]$ which is already constructed (with valence $n=3$ or not).

In the first case, i.e. when one of the e_j^k is known, the coordinates of v^k can easily be computed from the Equation **(2)**. The formula can be written as:

$$\boxed{v^k = 4e_j^{k+1} - e_j^k - f_j^{k+1} - f_{j-1}^{k+1}}$$

where the coordinates of e_j^{k+1}, e_j^k, f_j^{k+1} and f_{j-1}^{k+1} are known.

In the second case, i.e. when no vertices e_j^k are constructed, we can construct v^k in the cases that we describe below:

If there is one face F_j, $j \in [\![0,2]\!]$ such that all vertices $f_i^k \in F_j$ are already constructed, then we can generate a new system from Equation **(1)** and **(2)** written for this face F_j and the two edge points of this face incident to v^k:

$$\begin{cases} 4f_j^{k+1} = v^k + e_j^k + \sum_{f_i^k \in F_j} f_i^k + e_{(j+1)[n]}^k \\ 4e_j^{k+1} = v^k + e_j^k + f_j^{k+1} + f_{(j-1)[n]}^{k+1} \\ 4e_{(j+1)[n]}^{k+1} = v^k + e_{(j+1)[n]}^k + f_j^{k+1} + f_{(j+1)[n]}^{k+1} \end{cases}$$

From this, the formula to construct v^k verifies:

$$\boxed{\begin{aligned} v^k = 4e_j^{k+1} - f_j^{k+1} - f_{(j-1)[n]}^{k+1} - 5f_j^{k+1} - f_{(j+1)[n]}^{k+1} \\ - f_{(j+1)[n]}^{k+1} + 4e_{(j+1)[n]}^{k+1} + \sum_{f_i^k \in F_j} f_i^k \end{aligned}}$$

The only case where we cannot find a formula for v^k is when there is not adjacent face at the vertex v^k such as its vertices except v^k are already constructed. So we stop here our investigations because these results are sufficient for almost all cases. Indeed, the Catmull-Clark scheme is generated to be applied on quadrilateral meshes with a regular valence $n=4$. Of course, it is extended to polygonal faces and extraordinary valences $n \neq 4$ in order to be able to subdivide a greater number of meshes with this subdivision scheme. Indeed, the initial condition (a quadrilateral mesh with valence $n=4$) is a too restrictive condition.

However, applying this scheme on a triangular mesh is not the goal because there are schemes which are better adapted to these meshes.

The cases for which these formulas are not enough are thus those where the valence is 3 for all mesh vertices as in the case of the box for example (the 8 vertices have 3 for valence). Indeed, even in these kinds of cases, among the new inserted vertices, edge points have a valence equal to 4. Thus, at the next level of subdivision, the mesh corresponds to a case described above.

Boundary vertices

With the same notation as in the part of section 2 on boundaries, one formula given by Bartels and Samavati in [Bar00] is the following:

$$v^k = -\frac{1}{2}e_0^{k+1} + 2v^{k+1} - \frac{1}{2}e_1^{k+1}$$

This formula can exactly be used as the Equation **(4)** we give for the interior vertices with valence $n \neq 3$.

4. RESULTS

In this section, the method described in the previous section is applied to on meshes with different features.

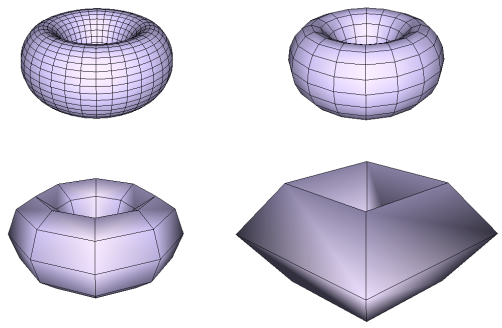


Figure 8. Reverse Catmull-Clark subdivision applied to on the torus mesh subdivided 3 times.

The easier example consists in rebuilding the torus of Figure 3 from one of its subdivision: the mesh at the third level of subdivision. As the initial mesh consists of quadrilateral faces with vertices of valence 4, all the new vertices are also of valence 4. The successive meshes can so be constructed with the general method described in the previous section (Figure 8).

This second example illustrates a case with extraordinary valences. There are valences equal to 3, 4 and 6. Vertices with valence 4 or 6 are constructed from the general method and vertices with valence 3 from the first formula introduced for valence equal to 3 because there is always one of the incident vertices with a valence not equal to 3. Successive meshes are shown in Figure 9.

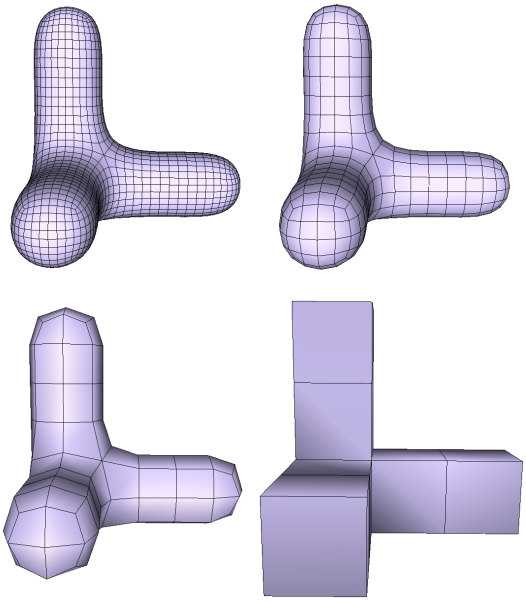


Figure 9. Reverse Catmull-Clark subdivision applied on a mesh with valences equal to 3, 4 or 6.

Let us consider the cat mesh. This mesh is triangular with arbitrary valences except the 3 value. Thus the general method can be applied to reconstruct the initial level from the first subdivision as shown in Figure 10. Indeed, face points are only vertices with valence 3 at the first subdivision level; they are marked with triangles in Figure 11. By construction, face points are computed as the centroid of a face, so there are no points which correspond to face points at the previous level but a face.

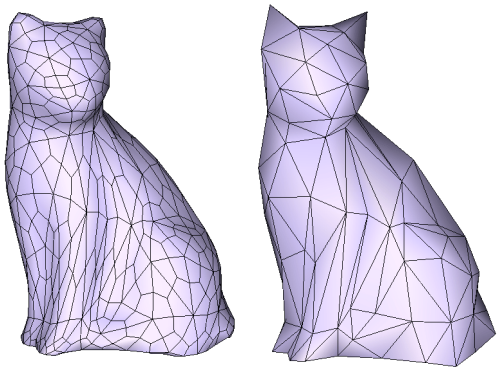


Figure 10. Reverse subdivision of the cat mesh at the first subdivision level.

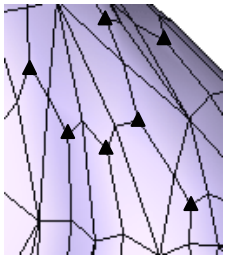


Figure 11. Zoom on a part of the mesh cat. Face points are represented by triangles.

If the mesh is subdivided once more (2 subdivisions), we have to construct vertices from vertices with valence 3 as illustrated in Figure 12.

Figure 13 focus on a part of the mesh to show what happens. Only the vertices to reconstruct are marked: those with valence 3 are represented by circles and the other by squares. In this mesh, there is always a vertex with a valence not equal to 3 between two vertices of valence 3. So the vertices of the previous mesh can be constructed from the general method and the first formula introduced for valence equal to 3.

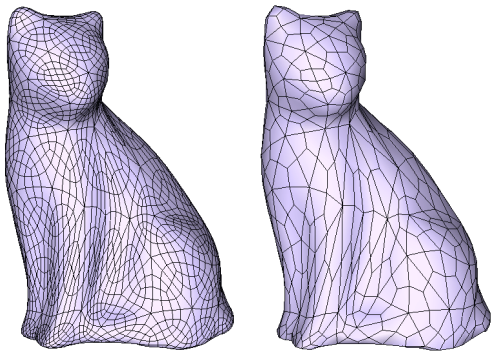


Figure 12. Reverse subdivision of the cat mesh at the second subdivision level.

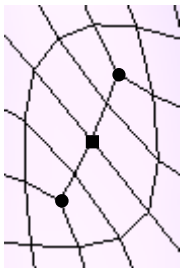


Figure 13. Zoom on a part of the mesh cat. Circles represent vertices with valence 3 and squares represent vertices with valence 4.

Let us consider a cube. At the initial level, all faces are quadrilateral and valence of vertices is always 3 (Figure 14 left). From the first level of subdivision, formulas explained here do not allow to determine vertices of the initial level from the first subdivision level because the vertices to construct all have 3 for valence (Figure 14 right). The same problem occurs with a tetrahedron.

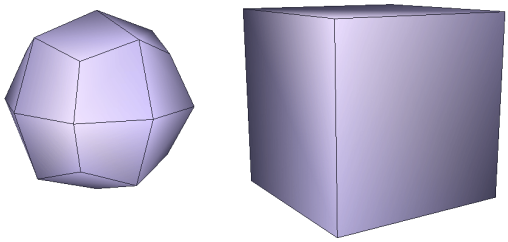


Figure 14. One particular example where the method fails: the cube starting from the first level of subdivision.

However, from the other successive subdivided meshes (except the first subdivision level), meshes of the previous level can be constructed from the general method and the first formula introduced for valence equal to 3. Indeed, vertices with valence 3 are isolated. Thus, Figure 15 shows on the left the cube mesh subdivided twice and its reconstructed mesh at the previous level.

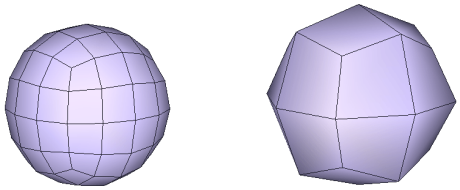


Figure 15. Reverse subdivision of the cube mesh at the second subdivision level.

5. CONCLUSION

With this method, coarser meshes can be quickly computed in almost all cases, with a local reverse subdivision mask. As for most reverse subdivision processes [Mon04], the locality allows very quick reverse subdivision. However, the coarser mesh can be found only if the current mesh was generated by subdivision. Indeed, if the mesh is not generated by refinement, the points (from vertex v^{k+1}, edge e^{k+1} or face f^{k+1}) are not distinguishable from each other so the application of the reverse process can be found only in particular cases. One reason is that the number of vertices is determined by the applied subdivision. This drawback can be found in any reverse subdivision process whatever the mesh (triangular, quadrilateral…) and whatever the subdivision rule (Loop, Doo-Sabin, Catmull-Clark…).

6. REFERENCES

[Bar00] Bartels, R., Samavati, F., Reversing Subdivision Rules: Local Linear Conditions and Observations on Inner Products in the *Journal of Computational and Applied Mathematics*, **119**(1-2), pp. 29-67, July, 2000.

[Cat78] Catmull E. and Clark J., *Recursively generated B-spline surfaces on arbitrary topological meshes.* Computer Aided Design, **9**(6), pp. 350-355, 1978.

[Cha74] Chaikin G.. "An algorithm for High Speed Curve Generation." CGIP 3, p. 346 – 349, 1974.

[Doo78] Doo D. and Sabin M., *Behaviour of recursive subdivision surfaces near extraordinary points.* Computer Aided Design, **9**(6), p. 356-360, 1978.

[Dyn90] Dyn, N., Levin, D. and Micchelli, CA, *Using parameters to increase smoothness of curves and surfaces generated by subdivision.* Computer Aided Design, **7**(2), pp. 129-140, 1990.

[Loo87] Loop C., *Smooth Subdivision Surfaces Based on Triangles*. Department of Mathematics: Master's thesis, University of Utah, 1987.

[Lou97] Lounsbery, M., DeRose, T.D.and Warren, J., Multiresolution analysis for surfaces of arbitrary topological type. ACM Transactions On Graphics, **16**(1), pp. 34-73, 1997.

[Mon04] Mongkolnam, P., Razdan, A., and Farin G., *Reverse Engineering Using Loop Subdivision* Computer-Aided Design & Applications, **1**(1-4), pp 619-626, 2004.

[Sam98] Samavati F. and Bartels R.. Multiresolution curve and surface representation: Reversing subdivision rules by least-squares data fitting. Computer Graphics Forum, **18**(2), pp. 97-119, 1998.

[Sam02] Samavati, F. and Bartels, H.. Reversing subdivision using local linear conditions : Generating multiresolutions on regular triangular meshes. Technical report 2002-711-14, December 2, 2002.

[Sam02] Samavati, F., Mahdavi-Amiri, N. and Bartels R.. Multiresolution surfaces having arbitrary topologies by a reverse Doo subdivision method. Computer Graphics Forum, **21**(2), pp. 121-136, 2002.

[Sam03] Samavati, F.,. Pakdel, H. R, Smith, C. and Prusinkiewicz, P.. *Reverse Loop Subdivision*. Technical report 2003-730-33, November 4, 2003.

Dynamic Annotation of Interactive Environments using Object-Integrated Billboards

Stefan Maass
University of Potsdam
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Strasse 2-3
14482 Potsdam, Germany
maass@hpi.uni-potsdam.de

Jürgen Döllner
University of Potsdam
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Strasse 2-3
14482 Potsdam, Germany
doellner@hpi.uni-potsdam.de

ABSTRACT

We present a technique for the dynamic annotation of three-dimensional objects in interactive virtual environments. Annotations represent textual or symbolic descriptions providing explanatory or thematic information associated with scene objects. In contrast to techniques that treat annotations as two-dimensional view-plane elements, our technique models annotations as separate three-dimensional scene elements that are automatically positioned and oriented according to the shape of the referenced object. The shape of such an object is generalized by an annotation hull and skeleton used to determine an adequate position and orientation of the annotation with respect to the viewing direction. During camera movements, annotations float along the surface of the annotation hull. Additional constraints for the generalizations provide further control about geometric and dynamical properties. In a case study, we show how this technique can be applied for annotating buildings and other components of virtual 3D city models.

Keywords

Annotation, labeling, real-time rendering, visualization, user interfaces.

1. INTRODUCTION

Annotation techniques differentiate between *internal annotations*, overlaying the referenced objects, and *external annotations* that are placed nearby the objects and connected with additional elements, mostly lines and arcs. An internal annotation partially obscures the referenced object while directly establishing a mental link between both parts. External annotations are preferably used to annotate small objects as well as large numbers of objects, or to group objects spatially by a specific topic. Here, the user follows the connecting element to associate the annotation with the referenced object. Crossings or long distances between the object and the annotation, therefore, should be avoided.

FULL Papers conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

There is a long tradition of annotating two-dimensional graphics, such as medical, botanical, or geological illustrations and geographical maps. Today, most techniques still lay out annotations in a two-dimensional manner even for interactive virtual 3D environments. Annotations are added after the scene elements are projected, that is, annotations are handled as 2D view-plane elements. This approach is referred to as view management and reduces the complexity for the arrangement by simplifying the annotation layout task to a two-dimensional placement problem. It has the advantage that these annotations parallel to the view-plane provide optimal readability.

However, whether an annotation technique is adequate or not depends to a large degree on the application context. If the user wants to inspect a single object from a bird's-eye view, the object can be centered in the view such that there remains enough white space around in which external labels can be placed. Elements can even be overlaid with textual descriptions if the visible amount of those parts is large enough. In contrast, if the user is immersed into a virtual environment (e.g., simulators for virtual

buildings and cities) the annotated objects surround the user. External annotations turn out to be problematic in this case because the white space for them needs to be detected, and this can only be solved at a semantic level. Even if a visible part of an object is large enough for an internal annotation, size, position, and number of these regions can completely change during the user movements, so they have to disappear or temporally overlay other objects.
In this contribution, we introduce the concept of *object-integrated* annotations, which are represented by separate 3D scene elements that keep dynamically attached to the referenced objects. They do not lose the visual link to the referenced objects and communicate their scope. We have implemented the approach in a prototypic system and applied for virtual 3D city models. In the following, we refer to *objects* of virtual environments or, if more specifically, to *buildings*.

2. RELATED WORK

For two-dimensional images, annotations can be implemented by integrating textual descriptions into the referenced image parts [Chi01a]. This concept of "dual use of image space" addresses the important problem of sharing space between image and text explanations and "achieves a smooth transition between the representation of an object as an image and its representation as a text" [Chi02a] to improve readability. For this Chigona and Strothotte develop a morphing technique that transforms a selected object into a rectangular window. In a sense, we extend this work to a "dual use of 3D space"; however, we do not distort 3D scene geometry.

In cartography, static label placement has been investigated yet for a long time, where typically text is integrated into the 2D maps; for a survey of algorithms see [Chr95a]. To achieve a high quality annotation placement, criteria such as disambiguation, selectivity, and expressivity of annotations are optimized [Har04a][Ebn03a][Edm96a].

Some approaches optimize these criteria with force-based algorithms [Ebn03a][Har04a]. The annotations are placed at initial positions on the view plane. Additional attracting and repulsive forces are defined among each other and the border of the view. Over multiple iterations, a relaxation process minimizes the overall forces, so that the annotations obtain improved positions. The computational costs do not allow for real-time label placement and, therefore, needs to be performed in a post-processing step.

The third dimension drastically increases the complexity of the label placement problem, both conceptually and algorithmically. 3D visualizations also go beyond classical map-based representations and demand different kinds of, and dynamic, labeling techniques. A first approach for 3D label placement is presented by Preim et al. [Pre97a]; their technique reserves part of the view plane for fixed containers for the textual descriptions linked by lines to the referenced objects. Ritter et al. [Rit03a] introduce the concept of illustrative shadows: Annotations are linked to reference points in the shadows of objects projected onto an additional shadow plane and, thereby, support an intuitive visual association. If there is enough white space between the shadows on the shadow plane and the object, the annotation placement can be reduced to a two-dimensional problem. Sonnet et al. [Son04a] investigate annotations in the context of interactive explosion diagrams intended to explore complex 3D objects.

The immersion of the user into annotated virtual environments (e.g., virtual pedestrian navigation) extends the degrees of freedom for annotation placement and partly the requirements for the annotation strategy. For example, for way finding in 3D city models annotations can occur that do not refer to visible objects. These annotations indicate nearby streets, places, or buildings, e.g., service stations, pharmacies, or hospitals. In addition, in immersive scenarios it is generally not desirable to depict all annotations; only a subset is determined that contains annotations referencing currently spatially near-by objects or objects relevant to a specific task. Kolbe investigates the annotation of buildings in pre-recorded path videos [Kol04a]. To calculate the placement of annotations an external 3D city model is required. The annotations augment the geo-

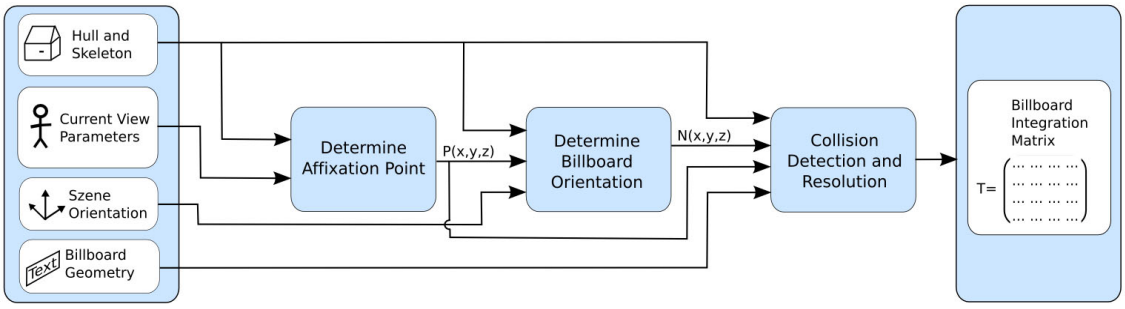


Figure 1: Overview over the annotation process.

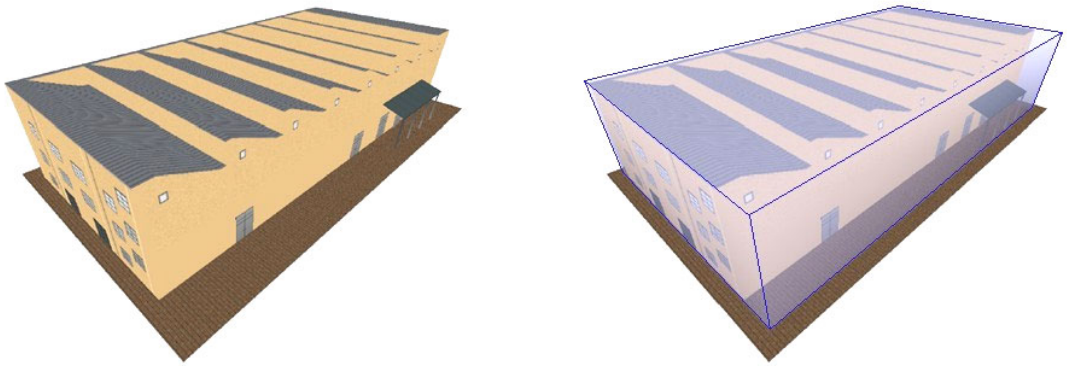


Figure 2: Factory building with a complex roof structure and a porch, with and without annotation hull.

referenced frame sequence of the video.

The work of Bell et al. [Bel01a] develops a technique for dynamically annotating 3D buildings by labels placed on the view plane based on the upright rectangular extent of object projections. Their viewing management system resolves visibility aspects and automates the switch between internal and external annotations. In our approach, in contrast, we integrate annotations in the 3D scene geometry, i.e., labels are not constrained to be parallel to the view plane.

3. OBJECT-INTEGRATED LABELS

In our approach, annotations are represented by 3D scene elements and, therefore, form part of the 3D scene geometry. An annotation is modeled by a textured billboard. The design of the annotation, i.e., its contents, is completely defined by the texture data. This way, we obtain a high degree of freedom for the graphical design of annotations; for the most common case of textual labels, corresponding textures can be created automatically. We can also specify multiple texture variants for different levels of detail, which are selected depending on the distance of the viewer.

The process of placing and integrating of an annotation can be divided into three steps (Figure 1). First, a fixation point at the referenced object is determined for the center of the billboard. Next, the billboard is orientated so that it appears to be integrated into the object. Because this can cause collisions between the annotation and the object, in the final step we detect and, if required, resolve such collisions.

3.1 Hull and Skeleton

To position the annotation our algorithm requires two *generalized variants of the annotated object*, a hull and a skeleton.

The hull represents the annotated object by a generalized geometry having a lower polygonal resolution. To generate a hull, we can take a low level-of-detail variant of an 3D object (e.g., progressive meshes) or derive the hull by object-specific techniques. For example, the hull of a 3D building model can be derived by eliminating small geometric details of its footprint and extruding the simplified footprint to its maximum height.

The hull is required to reduce the influence of detail geometry (e.g. in the case of 3D buildings: balconies, oriels, protrusions, etc.) that would cause a disturbed dynamic annotation placement. At the same time this decreases the computational costs in the case of geometric complex models. Figure 2 shows an example where a simple box is used as an annotation hull for a factory building with a complex roof structure and a protrusion.

The skeleton represents an internal supporting structure of a 3D object. For example, the skeleton of a 3D sphere can be represented by a single 3D point whereas the skeleton of a 3D torus can be represented by a 3D circle. For our approach, we require the straight skeleton, a collection of line segments that indicate the medial axes of the annotated object [Aic95a][Fel89a].

The skeleton is used to find a fixation point for the annotation. In particular, it is useful for those objects having a hull shape that is more complex than a sphere or a cube (e.g., buildings with non-simply shaped floor plan).

3.2 Affixation of Annotations

We propose two techniques for the calculation of the affixation point of the annotation in the 3D space. In the first variant a ray is sent from the viewer to the center of the annotation hull. The first ray intersection with the hull is used as the affixation point for the annotation. For simple object types (e.g., nearly

cubical buildings in our case study) this affixation strategy is efficient.

For complex 3D objects, we can extend the affixation strategy. For example, in the case of a building with a complex floor plan, we create a skeleton that represents the structure of the floor plan. Techniques to create such skeletons are known from the automatic creation of roof geometries [Aic95a][Fel98a] and can be transferred to other object types as well. To determine the affixation point first the point at the

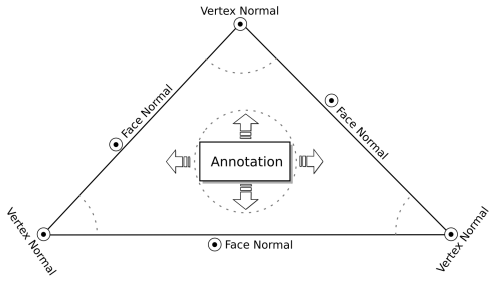


Figure 3: Normals of adjacent vertices and faces impact the resulting orientation, when the annotation affects their range of influence.

skeleton is calculated that has a minimal distance to the ray in the center of the observer's view. If this point is known, a ray will be sent from the observer's view. The destination is the calculated point on the skeleton instead of the hull's center.

3.3 Orientation

The degrees of freedom for the annotations after the affixation permit only an orientation by a rotation around the affixation point. To find an orientation we use the vertex and face normals of the hull from the polygons that are near the polygon containing the affixation point. If the annotation can be completely integrated into this polygon of the hull without overlapping an edge, the orientation is equal to the face normal:

$$\vec{n}_o = \vec{n}_{affixation}^{face}$$

At hard edges the labels should smoothly disconnect from the current object face (e.g., a facade of a building), rotate around the edge, and smoothly integrate again into the adjacent polygon. For this, the normals at the vertices and adjacent faces must be taken into account as well (Figure 3).

The orientation of the label is influenced by a vertex normal if the distance d_i from the label center to the vertex is below the radius d_{max}, defined by a half of the label's diagonal length. In this case the orientation is calculated by

$$\vec{n}_o = \sum_{i=0}^{\#vertices} (d_{\max} - d_i) \cdot \vec{n}_i^{vertex} + d_i \cdot \vec{n}_m ,$$

where the second interpolation argument ($\vec{n}_m$) is the result of an interpolation in itself. For this the proportion of the angle between the adjacent polygon edges weights the face normals in clock and counter clockwise order around the vertex:

$$\vec{n}_m = ((1 - w_a) \cdot \vec{n}_{ccw}^{face} + w_a \cdot \vec{n}_{cw}^{face})$$

If the label is not near a vertex, but near an edge of the polygon, the orientation is determined in a slightly different way. For every edge with a distance below d_{max} we interpolate between the face normal of the adjacent polygon and the face normal of the polygon containing the affixation point as follows:

$$\vec{n} = \sum_{j=0}^{\#faces} (d_{\max} - d_j) \cdot \vec{n}_j^{face} + d_j \cdot \vec{n}_{affixation}^{face}$$

Afterwards, the annotation is oriented so that the face normal is pointing in the same direction as the calculated normal. The result of these calculations is demonstrated in Figure 4, where these orientation normals are calculated in discreet steps over the hull.

3.4 Collisions

The affixed annotation can intersect the object so that it is partially occluded after it has been oriented (Figure 6). To avoid this, an additional step must detect and resolve such collisions.

3.4.1 Image Space

A simple screen-space technique can be used to

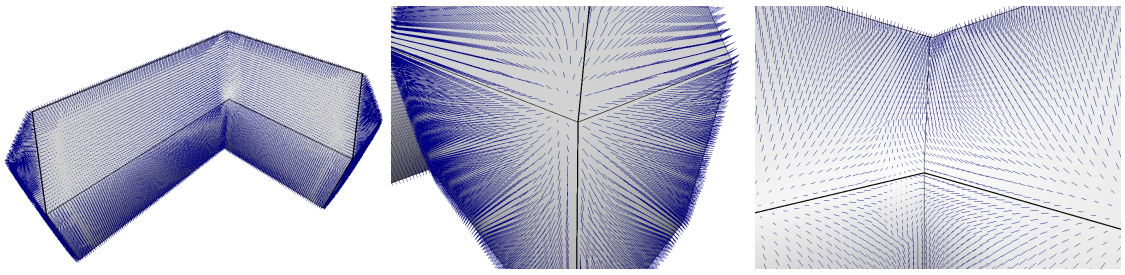


Figure 4: Normals, used to orient the annotation, calculated at discrete samples of the hull (overview, detailed view on a convex and concave edge).

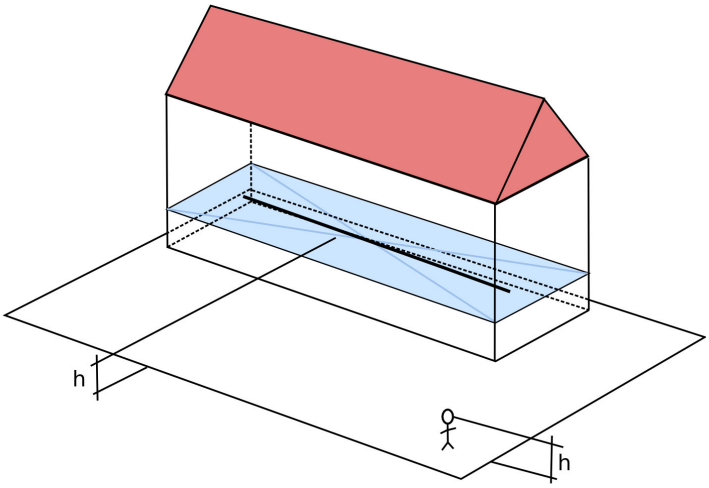


Figure 5: Affixation of the skeleton at the viewer's eye level instead in the center of the object.

avoid the occlusion of an annotation from the referenced object but requires a sorting of all scene elements (without annotations) by their distance to the observer. Starting with the furthest object all elements are drawn in this ordering [Fuc80a]. If this process comes upon an annotated scene element, the object is drawn first. After this the assigned annotation is rendered but with disabled depth-buffer test. As a consequence, it is drawn over the object, even if penetrating it.

3.4.2 Object Space

For scenes with nearby objects or higher geometric complexity an efficient and well-defined ordering is not always possible. This is the reason to develop a technique that detects and resolves the collisions in object-space.

For the annotation billboard we span a plane into the object space and check for possible intersections with the annotation hull. If these intersections are inside the borders of the billboard, it is moved using the direction of the orientation normal as a separating axis until the conflict is resolved.

For these tests only the polygons of the hull are used to reduce computational costs. If required, the intersection tests can be accelerated by making use of bounding volume hierarchies (e.g., [Got96a]) to reduce intersection tests to a few polygons only.

3.5 Annotation Selection

Using our approach in interactive applications with larger data sets raises the question of scalability. Dependent of the kind of application and the number of annotations, this question can be answered differently:

Thematic Limitation: In interactive applications the user can chose subsets of annotations that are to be displayed. These groups could be built from semantic information (e.g., landmarks, street names, building usage) in advance or created by a search request of the user (e.g., hints for way finding).

Restriction by Visibility Culling: We can reduce the number of annotations to be handled by visibility culling [Ass00a]. Only for visible objects or objects near the view frustum annotations are considered to be active. Additionally, the distance to the observer can be used to exclude annotations that are far away. During the transition between the visible and non-visible state the annotation's transparency is smoothly faded to avoid popping effects.

Delayed Annotation: We can give up or delay the interactive placement of annotations, that is, annotations stay fixed at their last positions until the interaction process comes to an idle state. Then, these annotations can directly be placed (with fade-in) at their new positions or can move along a path on the hull surface in order to allow the observer's eyes to follow them.

4. CONSTRAINTS

Constraints provide additional control how to place annotations in the 3D scene. We have worked out two constraint types so far.

4.1 Observer Height Constraint

Since the position of the skeleton inside the annota-

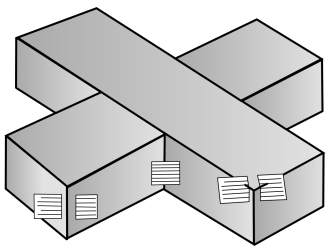


Figure 6: Different kinds of annotation-object collisions after the orientation pass of the process.

tion hull strongly affects the place of affixation, we can control the annotation placement by constraining the skeleton. Commonly, the skeleton is centered in the annotation hull to allow for an equal distribution of affixation points. Keeping the skeleton at the same height as the viewer's eye level (Figure 5) allows us to control the placement not only by the hull's shape but also by the viewer's position. As a result annotations tend to be more prominent in the user's view. For example, this type of constraint improves the placement in scenarios where the user interacts as a virtual pedestrian.

4.2 Object Height Constraint

A simple way to constrain the placement more by the shape of the hull and to limit it to only one dimension is to define a plane through an object. The position of the annotation can be restricted to the intersection line of this plane with the annotation hull. For this, the skeleton and the position of the viewer are projected down to the plane before the ray intersection between the viewer and the hull is calculated. As an example, for buildings this can be used to limit the annotation placement at the facade to the white space between the windows of two adjacent floors.

To restrict the placement area to a horizontal band of the object surface, a second plane, parallel to the first one, can be defined. If the observer is located between these planes the algorithm without any modifications is used, if not the position is projected to the nearest plane as described before.

5. RESULTS

We have tested our annotation technique with a set of individual objects and a 3D city model of our university campus (Figure 7, 8). The annotations have a defined background color and, therefore, provide a good compromise between high perception quality and seamless object-space integration. Since these annotations can occlude parts of the objects they can be rendered transparently.

In comparison to two-dimensional, view plane parallel annotations, the object-space integration of the annotations reduces the readability. This is uncritical in immersive scenarios where the user is navigating through the information space. The users realize the annotations as an additional information source and can easily obtain a position with better readability.

In our future work, we will develop additional quality criteria (e.g., degree of object-integration, quality of the integration area) and relate them to existing conditions (e.g. visibility, legibility). As a first step, a visibility determination feedback loop could be integrated into the process that optimizes the annotation position along the visible part of the hull.

A strength of the dynamic object-space annotations is that they allow for an exact identification of the annotated object parts. Unlike the additional lines of external labels or two-dimensional overlays of internal labels, the object-integrated labels communicate their areas of validity. Through this, in our case study the user was able to distinguish between the cases where the whole building, only a part, or a facade is referenced by the annotation.

Another benefit of object-integrated annotations is the high utilization of the available screen space – an aspect important for mobile navigation assistants having small displays.

While the user navigates through the 3D scene, the annotations are floating dynamically over the object

Figure 7: User study with multiple objects: annotated buildings of a campus model.

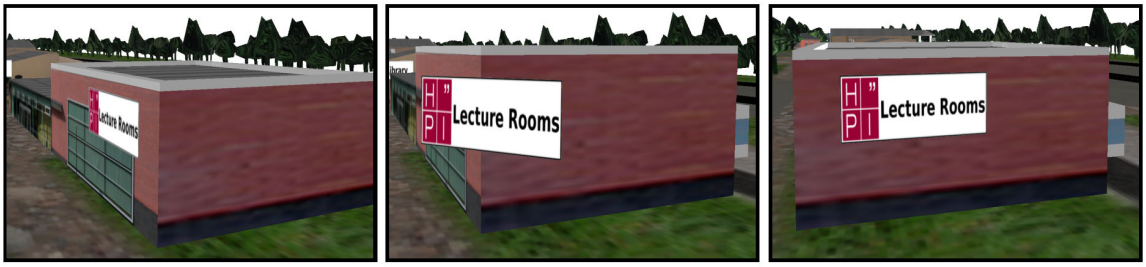


Figure 8: Sequence, showing an annotation detaching from one facade, rotate around the edge an integrate into the adjacent facade.

surfaces. Using a skeleton with more than one segment can sometimes cause some unsteady jumps, if the part changes that contain the point nearest to the view ray. An interpolation over the hull surface could solve this problem but requires additional calculations to plan the path between the last and the new position.

Because the geometry of the annotation hull controls the affixation and orientation of the annotation, the annotation creator gets degrees of freedom for designing this hull. For example, a more rounded hull could cover buildings with a lot of hard edges near by each other but if the difference between the hull and the object gets over a considerable threshold, the quality of the correlation will suffer.

Object-integrated annotations are suited for scenarios where the user is in the role of a virtual pedestrian. The annotations are perceived as commonly known plates, with the only difference that they are moving. The benefits decrease more and more if the user comes to a bird's eye perspective. In our test environment, especially buildings with flat roofs turned out to be problematic because the annoations get integrated rapidly into the roofs and, therefore, cannot be seen anymore as outstanding labels. Partially, this can be corrected with an adjustment at the hull. A more general solution should look for a transformation to another annotation technique, if the user is leaving the immersive perspective.

6. CONCLUSIONS

The concept of object-integrated annotations provides a solution to enrich 3D objects with thematic, textual or symbolic information within an immerse view to a virtual environment. Compared to view-plane parallel annotations, object-integrated annotations communicate more precisely their areas of validity in interactive applications. Additionally, they reduce the irritating situation that user interaction in a 3D environment results in 3D visual feedbacks from the scene together with 2D repositioning responses from the annotations.

Our approach allows for an easy integration of annotations into 3D models with a time-coherent movement during user interaction. The interactive behavior can be simply controlled by the explicit design of the hull, the skeleton, and additional constraints.

In our future work, we want to complete the implementation by an automatic generalization technique that extracts a first variant of the hull and the skeleton from the original models. Furthermore, level of detail techniques can be integrated at different points of the approach to improve the display of the annotations.

For the extension to non-immersive environments, more attention must be paid to the occlusion of annotations among each other and by other objects. Moreover, our concept can be complemented by a non object-integrated annotation technique. The transition between different annotation techniques is another research topic we plan to study.

Concerning form and placement, the buildings in our case study show that objects, where planar areas are predominating, are especially suited for embedding billboards. Here, the annotations are directly mounted at the facades, resulting in a good correlation between annotation and object.

ACKNOWLEDGMENTS

This research is partially supported by the EU research and technology development project TNT (www.the-neanderthal-tools.org/).

We thank our anonymous reviewers for their helpful comments and suggestions.

REFERENCES

[Aic95a] Aichholzer, O., Aurenhammer, F., Alberts, D., Gartner B. A Novel Type of Skeleton for Polygons. Journal of Universal Computer Science, 1(12): 752-761, 1995.

[Ass00a] Assarsson, U., Möller, T. Optimized View Frustum Culling Algorithms for Bounding Boxes. Journal of Graphic Tools, 5:1, Jan., 9-22, 2000.

[Bel01a] Bell, B., Feiner, St., Höllerer, T. View Management for Virtual and Augmented Reality. Proceedings of the 14th Anual ACM Symposium in User Interface Software and Technology, 101-110, 2001.

[Chi01a] Chigona, W., Schlechtweg, S., Thomas, S. Dual Use of Image Space: The Challenges of Explaining Visualizations from Within. In Schulze, T., Schlechtweg, S., Hinz, V. (eds). Simulation und Visualisierung 2001, SCS-Society for Computer Simulation Int., Delft, Belgium, 175-185, 2001.

[Chi02a] Chigona, W., Strothotte, T. Distortion for Readability of Contextualized Text Explanations. Proceedings of IEEE International Information Visualization 2002, London, 289-294, 2002.

[Chr95a] Christensen, J., Marks, J., Shieber, S. An Empirical Study of Algorithms for Point-Feature Label Placement. ACM Trans. Graph. 14, 3 (Jul. 1995), 203-232, 1995.

[Ebn03a] Ebner, D., Klau, W.K., Weiskircher, R. Force-Based Label Number Maximization, Technical Report TR-186-1-03-02, Technische Universiät Wien, June 2003, available at http://www.apm.tuwien.ac.at/publications/bib/pdf/ebner-03.pdf

[Edm96a] Edmondson, S., Christensen, J., Marks, J., Shieber, S. M. A general cartographic labeling algorithm. TR1996-004, Cartographica 33, 4, 13-23, 1996, available at http://www.merl.com/publications/TR1996-004/

[Fel98a] Felkel, P., Obdrmalek, S. Straight Skeleton Implementation. Proceedings of the 14th Spring Conference on Computer Graphics, Budmerice, Slovakia, 210-218, 1998.

[Fuc80a] Fuchs, H., Kedem, Z.M., and Naylor, B.F. On Visible Surface Generation by A Priori Tree Structures. Computer Graphics (Proceedings of ACM SIGGRAPH 80), 14(3), 124-133, 1980.

[Got96a] Gottschalk, S., Lin, M.C., Manocha, D. OBBTree: A Hierarchical Structure for Rapid Interference Detection, Computer Graphics (Proceedings of ACM SIGGRAPH 96), 171-180, 1996.

[Har04a] Hartmann, K., Ali, K., Strothotte, T. Floating Labels: Applying Dynamic Potential Fields for Label Layout, Lecture Notes in Computer Science, Volume 3031, Jan, 101-113, 2004.

[Kol04a] Kolbe, T. H. Augmented videos and panoramas for pedestrian navigation. Proceedings of the 2nd Symposium on Location Based Services and TeleCartography, 2004.

[Pre97a] Preim, B., Raab, A., Strothotte, T. Coherent Zooming of Illustrations with 3D-Graphics and Textual Labels. Proceedings. of Graphics Interface, 105-113, 1997.

[Rit03a] Ritter, F., Sonnet, H., Hartmann, K., Strothotte, T. Illustrative Shadows: Integrating 3D and 2D Information Displays. Proceedings of the 8th International Conference on Intelligent User Interfaces 2003, Miami, Florida, 166–173, 2003.

[Son04a] Sonnet, H., Carpendale, S., Strothotte, T. Integrating Expanding Annotations with a 3D Explosion Probe. Proceedings of the Working Conference on Advanced Visual Interfaces, Gallipoli, Italy, 2004.

An efficient continuous level of detail model for foliage

C. Rebollo
Universitat Jaume I,
Castellón, Spain
rebollo@uji.es

I. Remolar
Universitat Jaume I,
Castellón, Spain
remolar@uji.es

M. Chover
Universitat Jaume I,
Castellón, Spain
chover@uji.es

O. Ripollés
Universitat Jaume I,
Castellón, Spain
oripolle@uji.es

ABSTRACT

Outdoor scenes require vegetation to make them look realistic. Current hardware cannot afford real-time rendering of these scenes because of the large number of polygons. Multiresolution modelling has been successfully presented as a solution to the problem of efficient manipulation of highly detailed polygonal surfaces. This article describes a new continuous multiresolution hardware-oriented model that can represent tree foliage with different levels of detail. The multiresolution model presented in this paper, *Level of Detail Foliage*, takes advantage of the programmable rendering pipelines nowadays available in most video cards. The geometry of the foliage is divided into a number of clusters, in some of which the detail can change while the rest of the clusters remain unaltered. This division of the foliage remarkably diminishes the number of vertices sent to the graphics system because only the information of the changed clusters are updated. The independent clusters condition a data structure that makes the time required for visualisation of the foliage more efficient. Here we present the data structure and the retrieval algorithms, which favour the extraction of an appropriate level of detail for rendering.

Keywords: Tree visualisation, interactive visualisation, multiresolution modelling, level of detail, hardware-oriented data design, clustering.

1 INTRODUCTION

Many of the interactive applications currently available such as flight simulators, virtual reality environments or computer games take place in outdoor scenes. In these environments, the vegetation is one of the essential components. The lack of trees and plants can detract from their realism. Nowadays, research on representation of plants has made possible to obtain very realistic models formed by a vast amount of polygons (Figure 1). This is a drawback for obtaining real-time visualisation.

Several methods have appeared up to now to solve this problem. They can be classified in two main groups: image and geometry based rendering. The geometry-based approach do not lose realism even when the camera is extremely near the object. Besides, this approach makes it possible to take advantage of the current graphics hardware, obtaining shadows or different ilumination effects. Other advantage is that geometry can be stored either in the main memory or directly in the graphics hardware, producing great acceleration in rendering.

Some techniques have been used in order to achieve interactive visualisation of very detailed objects. Multiresolution modeling has proved to be a good method as it adapts the geometric detail of these objects to the capacity of present-day graphics systems. These models typically work with general meshes. Nevertheless, Remolar et al. [16] presented in their work a multiresolution method especially designed to deal with the isolated polygons that form the foliage. (Figure 2).

Figure 1: Tree generated with the Xfrog commercial tool [22].

In this article, it is presented an efficient implementation of this multiresolution model, called *Level of Detail Foliage* LoDF. Its data structure has been designed as hardware-oriented in order to obtain a fast visualisation of the data. Besides, LoDF allows us to change the uniform level of detail in a continuous way. The basic idea of LoDF is to group the leaves in a number of independent clusters. This division makes it possible to exploit the capabilities of the latest hardware. All the geometric data is initially stored in the graphics card. In every change of level of detail, only the information about

WSCG 2006 conference proceedings, ISBN 80-903100-7-9
WSCG'2006, January 30 – February 3, 2006
Plzen, Czech Republic.

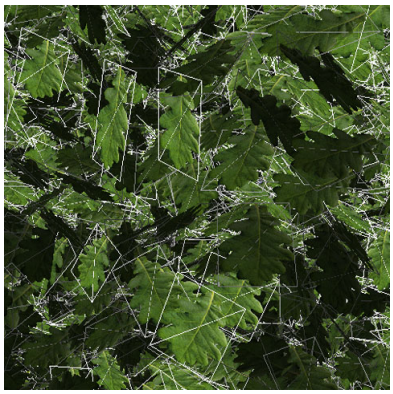


Figure 2: Detail of the foliage in a tree.

the clusters affected is updated and sent to the graphics system. The rest of clusters remain unchanged. This considerably reduces the traffic of data through the PCI bus and accelerates the visualisation process.

The article presents the following structure. Section 2 offers a brief review of the most important works in the visualisation models that have appeared up to now. In section 3 LoDF is analysed in depth, detailing the data structures it uses to store the information and the algorithms required to access these structures in order to retrieve that information in the most efficient manner. The visualisation process is proposed in section 4. Section 5 compares the results obtained with LoDF against the obtained using the model *View-Dependent Foliage* VDF presented by Remolar et al. [16]. Lastly, section 6 presents the conclusions and our future lines of work are outlined.

2 RELATED WORK

Research into real time visualisation of detailed plants is aimed at adapting the number of polygons used to represent the plants to the requirements of graphics hardware. As it has been previously said, these works can be grouped into two broad directions, depending on the method used to represent them. These can be works that use images or works that use geometry to represent the plants.

Image-based rendering is one of the most popular methods to represent trees because of its simplicity. Impostors [8] are the most common example of this approach. All the geometry that forms the tree model is previously rendered so as to obtain an image of it. This image is textured on a polygon using transparencies for correct immersion in the scene by replacing the geometric object. This way of representing vegetation has many problems, like the lack of parallax or the invariability of the rendered image when the camera changes its position. Some authors, such as Shade et al. [17], Marshall et al. [9] and House et al. [5], divide the scene into zones depending on the distance from the object to the viewer. Objects far away from the camera are represented by an image and objects near the viewer are depicted by geometry. Max [10] and Shade et al. [18] add depth information to the precalculated images. Other authors obtain 2D images from volumetric textures and combine them depending on the position of the camera. Meyer et al. [11] and Decaudin et al. [1] focused their work on forest visualisation, while Jakulin [6] and Reche et al. [13] were more interested in the representation of a single tree. Garcia et al. [3] solve the parallax problem using textures that group sets of leaves. One of most important commercial applications that uses this method to visualise trees is SpeedTree [19], which uses *billboards*, that is, impostors that are always oriented towards the viewer.

Regarding geometry-based rendering, the number of polygons that form the tree objects makes it necessary to use certain techniques to obtain interactive visualisation. Most of the works published to date change the display primitive using points or lines. Reeves and Blau [14] use particle systems to render trees in a forest, representing them with little circles and segments of straight lines. Weber and Penn [21] introduce level of detail techniques. In their work, leaves are represented with points while branches and limbs are represented with lines. The number of primitives that are displayed can be adapted depending on the size of the tree in the final image. Stamminger et al. [20] use points to represent trees in their work, changing the point density in real time depending on the relevance of the tree in the scene. Deussen et al. [2] and Gilet et al. [4] combine point and line representation with geometric representation. Their basic idea is to keep constant the number of vertices used to represent the forest. Trees near the camera are represented by geometry and trees situated far away from it are shown by points and lines.

In recent years, several works based on multiresolution models have appeared. Meyer et al. [12] and Lluch et al. [7] use a representation based on multiresolution models of images while Remolar et al. [16] deals with the geometry of the trunk and the foliage separately. They present a multirresolution model that adapts the number of leaves that form the tree in real time.

3 MULTIRESOLUTION MODEL

In general, a multiresolution model representation is constructed from two main elements: the original geometry of the object F_0, and the different approximations given by a simplification method, $F_1, F_2, ..., F_{n-1}$. The multiresolution scheme presented in this article is based on the Foliage Simplification Algorithm (FSA) [15]. The simplification operation of this method is *leaf collapse*: two leaves are collapsed into a new one.

LoDF uses two operations in order to increase or decrease detail in the approximation: *leaf split* and *leaf collapse* (Figure 3). The leaf-collapse operation diminishes the detail of the current approximation because it

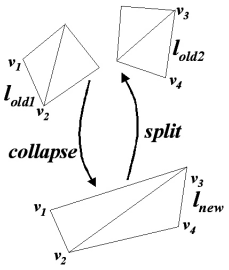


Figure 3: Example of leaf collapse and split operations.

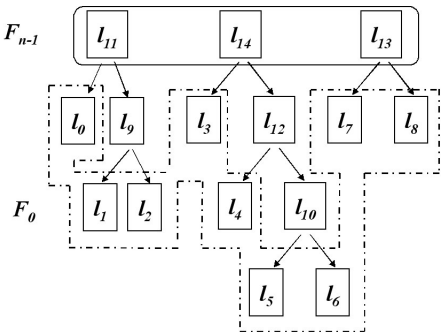


Figure 4: Example of data organisation in a F^r representation.

eliminates two leaves and adds a new one. In the example, leaves l_{old1} and l_{old2} will be eliminated and l_{new} added to the new approximation. However, the leaf-split operation adds detail to the representation being visualised. It eliminates one leaf l_{new} from the current approximation and adds the two leaves that were previously simplified, l_{old1} and l_{old2}.

$$Collapse(l_{old1}, l_{old2}) = l_{new} \quad (1)$$

$$Split(l_{new}) = (l_{old1}, l_{old2}) \quad (2)$$

The sequence of leaf-collapse operations obtained from FSA is processed to build the new multiresolution representation F^r. Each simplification operation creates one new leaf, numbered sequentially. In the case of a foliage with 9 leaves, labelled from l_0 to l_8, the first collapse operation creates the leaf l_9, the next simplification operation the leaf l_{10}, etc. One example of this data organisation is shown in Figure 4. The data are organised as a forest of binary trees, where the root-nodes are the leaves that form F_{n-1}, the coarsest approximation, and the leaf-nodes are the leaves of the original tree model, F_0. In this example, F_0 is formed by 9 leaves, and F_{n-1} by 3 leaves.

In the least detailed level F_{n-1}, several leaves from F_0 are represented by an only leaf. All of the leaves that form a binary tree in the data organisation are grouped in clusters in LoDF. In that way, foliage can be divided into independent groups. These clusters determine the data structure used in our model. Following the data structure shown in Figure 4, the groups that form this example are shown in Figure 5.

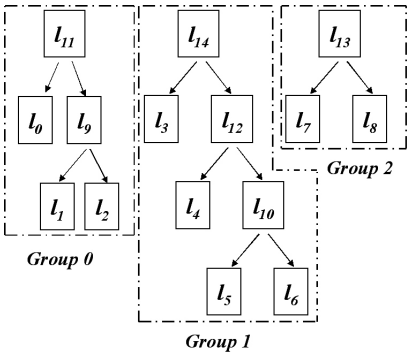


Figure 5: Organisation of Groups in F^r.

$l_{original}$	gr
l_0	0
l_1	0
l_2	0
l_3	1
l_4	1
l_5	1
l_6	1
l_7	2
l_8	2

l_{old1}	l_{old2}	l_{new}	gr
l_1	l_2	l_9	0
l_5	l_6	l_{10}	1
l_0	l_9	l_{11}	0
l_4	l_{10}	l_{12}	1
l_7	l_8	l_{13}	2
l_3	l_{12}	l_{14}	1

Figure 6: Information obtained in the preprocess. Every leaf and every collapse information is classified in one group.

3.1 Obtaining the data structure

First of all, it is necessary to determine the number of independent clusters that form the foliage. In this way, the information obtained from the simplification process FSA has to be processed. This simplification method provides the sequence of leaf-collapse operations to obtain F_{n-1} from the original model F_0. In order to build the multiresolution model, this sequence is processed in a previous process to obtain the number of the group every collapse concerns. It allows us to determine the number of final clusters in the foliage. When processing this information, every leaf in the foliage is assigned to one of the clusters. Regarding the example of Figure 4, the obtained data are shown in Figure 6.

Finally, this information is used to build the multiresolution data structure F^r.

3.2 Data Structure

All the leaves that form the foliage (not only the original ones but also those obtained in the simplification process) are stored in F^r. They are organised as an array of clusters, where all the leaves that form each group are stored together with some information enabling a

Init_group		0		0		0	
Active_leaves		3		4		2	
Changes_number		0		0		0	
leaf_number	next						
		l_1	1	l_5	1	l_7	1
		l_2	2	l_6	2	l_8	2
		l_0	3	l_4	4	l_{13}	-1
		l_9	4	l_{10}	5		
		l_{11}	-1	l_3	3		
				l_{12}	6		
				l_{14}	-1		

Figure 7: Example of data stored in the array of clusters that form F^r.

correct visualisation. In each cluster, leaves are stored traversing the binary tree by levels of depth. In Figure 7 is shown the data organization for the example of the Figure 6.

For every leaf in this array, it is stored the index of the leaf, the index of the vertices that form it and the position of the next leaf in the group to be visualised.

In addition, every cluster stores some information needed to obtain the appropriate sequence of leaves to be visualised in the required approximation:

Active_leaves. Stores the number of leaves to be visualised in the current level of detail.

Init_group. This information is used to store the position in the array of the first leaf to be visualised in the current approximation.

Changes_number. It contains the number of leaf collapse or split operations that have to be processed in the group to obtain the new level of detail required.

Data stored in Figure 7 are the ones of the original foliage F_0. All the clusters have as the *init_group* the 0 position. *Active_leaves* stores the number of leaves to visualise in each group, in this case, 3 leaves in the 0 group, and 4 and 2 for the next ones. In the example, the shaded fields represent the leaves in the current approximation. No changes has to be processed, so *changes_number* is initialized to 0.

Finally, it is necessary to store the sequence of leaf collapses obtained by the FSA. As a consequence of the data organization, only the number of the cluster where the simplification happens is necessary. So, following the example of the data structure of the Figure 4, it is stored the data shown in Figure 8. The field *init_position* indicates the position of the last leaf collapse operation performed. Initially, it is situated in the 0 position of the array.

This information allows us to change the detail of the approximation visualized in real-time.

gr	0	1	0	1	2	1

Figure 8: *Changed_group* structure. Information stored in order to achive an appropriate data retrieval.

Storage Cost Let F^r be an arbitrary LoDF representation; this stores the basic elements that compose a tree, that is to say, its vertices and polygons. Let $|V^r|$ and $|L^r|$ be the number of vertices and leaves stored in F^r, and V and L be the initial numbers of vertices and leaves that composed the crown of the tree. Note that in our multiresolution model:

$$|V^r| = V \tag{3}$$

since the simplification algorithm LoDF is based on, FSA, does not add new vertices when performing the leaf-collapse operation. Regarding the number of leaves, let L be the initial number of leaves,

$$L = \frac{V}{4} \tag{4}$$

since each leaf is formed by 4 vertices independent of the ones forming the other leaves. In every leaf-collapse operation, two leaves disappear and a new one is included. In the worst case, when the maximum number of leaf-collapse operations are carried out, $L-1$ new leaves are added to the initial L.

$$|L^r| = L + (L-1) = 2L - 1. \tag{5}$$

Let G the number of groups in the model. This number is conditioned by the number of leaves in the coarsest approximation F_{n-1}. In the worst case, when the model processes the maximum number of leaf collapse, F_{n-1} will be formed by a single leaf, that is to say, one group.

$$G = 1 \tag{6}$$

Let us suppose that the storage cost of an integer, real number or pointer is one word. The current implementation is not optimised for space, so each leaf would therefore have a cost of 6 words and each vertex 3. In this case, the main data structure of the model has a cost of:

$$3|V^r| + 6|L^r| + 3G \approx 3|V^r| + 6|L^r| \tag{7}$$

The number of leaf collapses conditions the size of the *Changed_group* structure. As said above, if we add $L-1$ new leaves, then this will be the number of leaf-collapse operations processed in our model. So, the size of *Changed_group* is $L-1$, i.e., L. Finally, the data structure cost is:

$$3|V^r| + 6|L^r| + L \tag{8}$$

Summarising, applying the equations 4 y 5, we can say that the storage cost of LoDF is O(V).

$$3V + 13V/4 \approx 6,25V \tag{9}$$

3.3 Retrieval Algorithms

The geometric data of the original model F_0 are initially stored in the graphics card.The application where the multiresolution representation of the foliage is used determines a level of detail depending on the relevance of the tree in the final image. The number of leaves that form the approximation can be changed in real time, adapting it to the requirements of the application. Once this number has been determined, two cases can happen:

- If the current number of leaves is greater than the determined by the application, some leaf collapse operations have to be performed. Considering that every collapse operation eliminates two leaves and adds a new one, it will be performed as many simplification operations as the number of leaves that have to be reduced in the new approximation.
- If the current number of leaves is less than the determined by the application, some leaf split operations have to be performed. Then, it will performed as many leaf split operations as the number of leaves have to be increased in the level of detail.

The sequence of simplification operations, as it is said in the previous section, is determined by the groups where each collapse happens (Figure 8). The level of detail of the current approximation can be changed traversing this information. Each time one simplification operation has to be performed in one cluster, it is increased its field *changes_number* in one.

Once this process is finished, all the clusters whose stored field *changes_number* is other than 0, has to be updated. In every affected group, some process has to be carried out.

Reducing detail. Each collapse operation will be as follows:

- *init_group* will be updated with the new position of the first leaf to be visualised,

$$init\_group + 2 \times changes\_number \tag{10}$$

  because in each leaf collapse two leaves disappear.
- the *active_leaves* will be updated with the new number of leaves to be visualised. The new value will be

$$active\_leaves - changes\_number \tag{11}$$

Init_group	4	2	0
Active_leaves	1	3	2
Changes_number	2	1	0

leaf_number	next					
l_1	1	l_5	1	l_7	1
l_2	2	l_6	2	l_8	2
l_0	3	l_4	4	l_{13}	-1
l_9	4	l_{10}	5		
l_{11}	-1	l_3	3		
		l_{12}	6		
		l_{14}	-1		

Figure 9: Data information after processing three leaf-collapse operations from F_0.

Increasing detail. For each split operation, the algorithm will involve the inverse process, that is to say:

- regarding the field *init_group*,

$$init\_group - 2 \times changes\_number \tag{12}$$

- and every split operation involves increasing the number of leaves to be visualised by one, so the field *active_leaves*

$$active\_leaves + changes\_number \tag{13}$$

Let follow the example of data shown in Figure 4 and the sequence of simplification operation shown in Figure 8. In the case of performing three leaf-collapse operations from the data organisation of Figure 7, according to the stored information, these would affect groups 0 and 1. New data structure will be as the one shown in Figure 9. The leaves to visualise in the current approximation are the shaded ones. Besides, the new value stored in *init_position* of the *Changed_group* data structure would be 3. The last process, once the affected groups have been updated, is to reset to 0 the *changes_number* field into all the groups.

4 VISUALISATION PROCESS

Every simplification operation only affects one group. This characteristic allows us to send only information about the group affected to the graphics card, instead of sending all the geometry of the foliage. Once the cluster affected is updated, the new leaves to be visualised will be the only information that will pass to the hardware. The rest of the groups will remain in the graphics card without modifications, that is to say, just as they were before the change. Thus, we will be updating the information in the card group by group. The model also offers the possibility of doing more than one simplification operation at a time and only those groups where some change has taken place will be sent to the hardware. This process reduces the visualisation time of a

tree considerably if we compare it with other methods that do not take advantage of the coherence property of the graphics card.

5 RESULTS

The method developed here was implemented with OpenGL on a PC with Windows XP operating system. The computer was a dual Pentium Xeon at 1.8GHz. with an NVIDIA GeForce 6800 Series GPU. The trees used in our study were modeled by the Xfrog application [22]. Results are shown in Figures 10 and 11.

In the experiments, the level of detail varies between 0 and 1, 0 being the most detailed approximation and 1 the least. In the figures it is shown how the number of leaves changes for each level of detail. The tests we carried out consist in traversing all levels of detail with a uniform *step*. This step represents the number of leaves that will appear or disappear in every approximation. It is proportional to the difference in the number of leaves between F_0 and F_{n-1}.

Results obtained using LoDF are compared with the ones obtained using VDF. Besides how the number of leaves changes, two graphs are offered in every figure, showing the results:

Total time. Time that the models spends on extracting and also visualising each level of detail.

Extraction time. Time that the models use to extract the necessary geometry to change from the most detailed approximation to the coarsest one.

In the figures it is easily seen how the presented model remarkably improves the results obtained with the VDF model. The hardware-oriented design of the structure accelerates the time employed in visualising a level of detail from a 63% in the case of the *Betula lenta* to a 88%, and in the case of the *Taxus baccata*. The extraction time is also reduced. It also depends on the number of leaves that form the foliage. The more leaves the foliage has the higher percentage of saving visualisation and extraction time it has. In the case of the *Betula lenta* this time is reduced in a 42% and in the case of the *Taxus baccata*, in 70%.

In Figure 12, some different levels of detail of the tree Carya ovata are shown. They vary in a uniform way from 114.114 to 28.528 leaves. Besides, the different approximations are composed following the distance to the viewer criterion. It can be observed how the representations maintain a high similarity with the original tree.

6 CONCLUSION AND FUTURE WORK

In this article we have presented a new multiresolution model that exploits the characteristics of current graphics hardware. This model, *Level of Detail Foliage* allows us to change the level of detail of a foliage representation in a continuous way. Its main advantage is that it tries to reduce the traffic of data through the AGP/PCIe bus diminishing the information that is sent to it when a change of level of detail is produced. This is obtained by grouping leaves in independent clusters and only modifying a small set of data.

This model allows us to represent forest scenes in interactive applications by instancig trees. It produces better results than current models based on images or points. Using geometry to visualise foliage makes it possible to render every detail of the tree. Another advantage of the geometry is that this can be adapted to the characteristics of the graphics hardware.

Another line of research we are currently working on is to obtain advanced ilumination effects and animation of the foliage. We are working on taking advantage of graphics hardware programming.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish Ministry of Science and Technology (TIN2004-07451-C03-03 and FIT-350101-2004-15), the European Union (IST-2-004363) and FEDER funds.

REFERENCES

[1] P. Decaudin and F. Neyret. Rendering forest scenes in real-time. In *Rendering Techniques '04 (Eurographics Symposium on Rendering)*, pages 93–102, 2004.

[2] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 219–226. IEEE Computer Society, 2002.

[3] I. Garcia, M. Sbert, and L. Szirmay-Kalos. Leaf cluster impostors for tree rendering with parallax. In *Proc. Eurographics 2005 (Short Presentations)*. Eurographics, 2005.

[4] G. Gilet, A. Meyer, and F. Neyret. Point-based rendering of trees. In P. Poulin E. Galin, editor, *Eurographics Workshop on Natural Phenomena*, 2005.

[5] D. House, G. Schmidt, S. Arvin, and M. Kitagawa-DeLeon. Visualizing a real forest. *IEEE Computer Graphics and Application*, 18(1):12–15, 1998.

[6] A. Jakulin. Interactive vegetation rendering with slicing and blending. In A. de Sousa and J.C. Torres, editors, *Proc. Eurographics 2000 (Short Presentations)*. Eurographics, 2000.

[7] J. Lluch, E. Camahort, and R. Vivo. An image based multiresolution model for interactive foliage rendering. *Journal of WSCG'04*, 12(3):507–514, 2004.

[8] P. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 95–102. ACM Press, 1995.

[9] D. Marshall, D. Fussell, and A.T. Campbell III. Multiresolution rendering of complex botanical scenes. In Wayne Davis, Marilyn Mantei, and Victor Klassen, editors, *Graphics Interface*, pages 97–104, 1997.

[10] N. Max. Hierarchical rendering of trees from precomputed multi-layer z-buffers. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96, Proceedings of the Eurographics Workshop*, pages 165–174. Eurographics, Springer-Verlag, 1996.

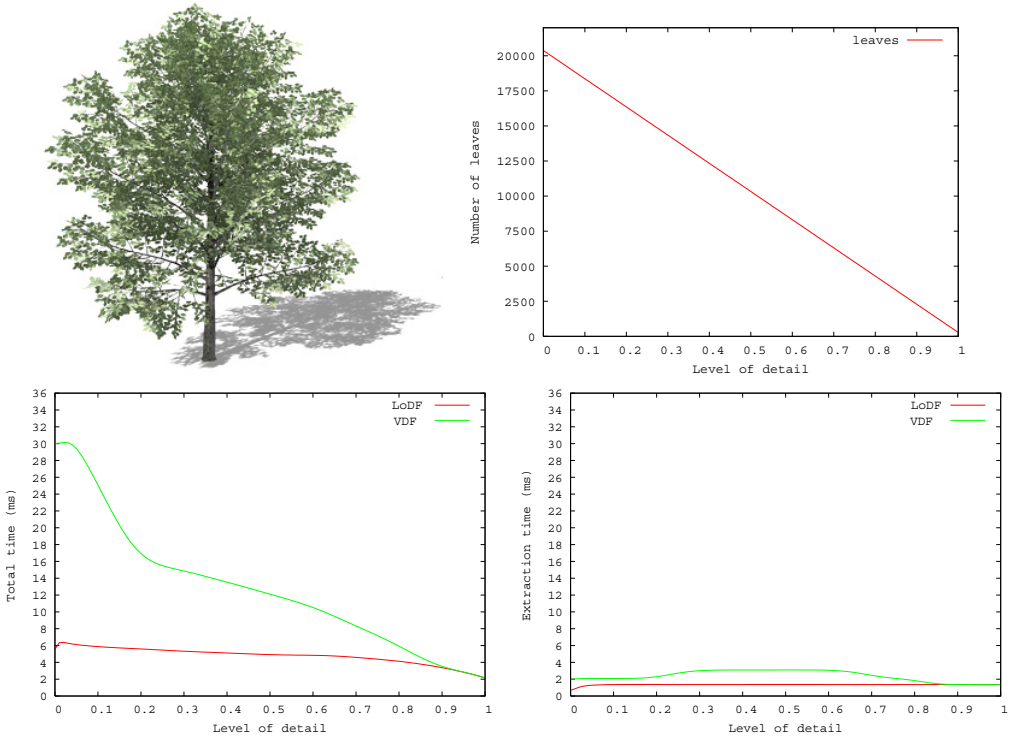


Figure 10: Results obtained for the tree *Betula lenta*.

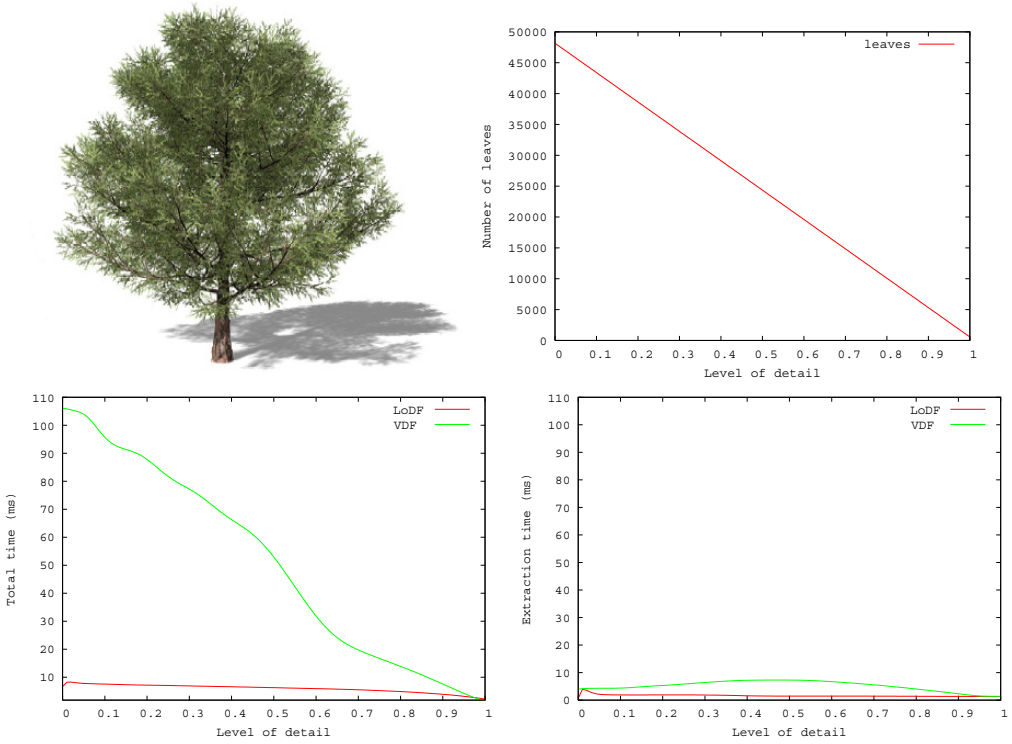


Figure 11: Results obtained for the tree *Taxus baccata*.

Figure 12: Above, from left to right: 114.114 leaves, 85.585 leaves, 57.057 leaves and 28.528 leaves. Below, composition of different aproximations following the distance to the viewer criterion.

[11] A. Meyer and F. Neyret. Interactive volumetric textures. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, pages 157–168. Springer Wein, 1998.

[12] A. Meyer, F. Neyret, and P. Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*. Springer-Verlag, 2001.

[13] A. Reche, I. Martin, and G. Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 23(3):720–727, 2004.

[14] W. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 313–322. ACM Press, 1985.

[15] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo. Geometric simplification of foliage. In I. Navazo and Ph. Slusallek, editors, *Proc. Eurographics 2002 (Short Presentations)*. Eurographics, 2002.

[16] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte. View-dependent multiresolution model for foliage. *Journal of WSCG'03*, 11(2):370–378, 2003.

[17] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 75–82. ACM Press, 1996.

[18] J. Shade, S.Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM Press, 1998.

[19] Speedtree, interactive data visualization inc. `http://www.idvinc.com/speedtree/`, 2005.

[20] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In S.Gortler and C.Myszkowski, editors, *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 151–162. Springer-Verlag, 2001.

[21] J. Weber and J. Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128. ACM Press, 1995.

[22] Greenworks: Organic software. `http://www.greenworks.de/`, 2005.

Visualizing Dynamic Etching in MEMS VR-CAD Tool

Renate Sitte
Griffith University
Faculty of Engineering and IT
Australia, Gold Coast Qld 9726
r.sitte@griffith.edu.au

Jie Cai
Griffith University
Faculty of Engineering and IT
Australia, Gold Coast Qld 9726
Jie.Cai@student.griffith.edu.au

ABSTRACT

In this paper we introduce our virtual etching as part of MAGDA a CAD system for Micro Electro Mechanical Systems (MEMS). Virtual prototyping visualizations require fast algorithms for visualization that are suitable for interactive design. Modern MEMS simulators do not offer dynamic visualizations for etching. Etching progress is time dependent, typically calculated with Finite Element Analysis, which has too slow a calculation time, hence is not suitable for interactive design. Etching progress is important in MEMS with small dimensions, where Silicon technology must be used, with its repeated cycles of deposition and lithography/etching until the desired structure is formed. While etching performance is well known from the Integrated Circuit processing, it is not so predictable in MEMS because the shapes are more complex. Underetching is not desired in IC technology, but it is crucial in shaping MEMS structures. We use a Marker/String method for the progressive mesh as a faster method suitable for interactive design. The method is not known much for etching; but used in other applications. We have found a way of overcoming swallowtail conditions that appear on corners. We are also able to simulate underetching. In this paper we demonstrate the progress of etching using a circular lithography mask calculated in 2D then rotated, and a square mask calculated in 3D. In both cases we are able to simulate underetching. The method can be extended into larger material removal CAD visualizations. In this way we made a step towards filling a long existing need in virtual prototyping.

Keywords

Scientific Visualization, VR-CAD tools, MEMS, Etching

1. INTRODUCTION

CAD tools have had an enormous impact in the design of any engineering product. When CAD tools are joined with virtual reality visualizations they provide invaluable visual feedback at the time of design. With improvements in computer speed and in computer graphics this has made drastic changes to the CAD design packages that are available. However, no matter how fast the hardware is, the problem of fast models for interactive design has not yet been overcome, and will always be a bottleneck.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

The problem arises because for dynamic CAD visualizations typically two phases are required. One phase involves preparation of the visualization as a sequence of frames, that is for example where the 3D change in the material being etched is calculated, followed by its rendering for a 2D screen. The other phase consists in playing the animated video clip. Together they are a lengthy process and unsuitable for interactive design.

Scientific visualizations including multi-dimensional multivariate visualizations have now been around for several decades, e.g. environmental maps of pluviosity. In our research, we go a step further, trying to display results of predictive calculations dynamically on the very design visualizations of the structures they represent thus adding to the information content they can offer. In particular we aim for fast models that are suitable for interactive CAD design. In this paper we present the dynamic process of etching. Our environment is in Micro-

Electro-Mechanical systems (MEMS) CAD development.

The introduction of CAD packages was a critical step in the widespread development of Integrated Circuits (IC) and reduction of the design and prototyping phase [Kar97]. There is a demand for CAD tools to aid in the development of MEMS devices. The typical evolution of CAD tools is that they emerge from applied research when particular devices were developed at different times, coming from specialized applications, rather than from specific design of the CAD tool. The result is a concoction of un-coupled and even incompatible pieces of software that are united under the umbrella of a "workbench". In such environments, computer crashes are common, leading to frustration and loss of time.

A rather small number of MEMS design software environments are available on the market. Perhaps the more widely advertised are ANSYS [ANS05] a multiphysics solver, Coventor [Cov05], Intellisense [Int05], and Femlab [Fem05] offering user-friendly Finite Element Analysis (FEA) environments and data mapped graphic results.

Other packages appear as a collection of tools at times limited to very specific applications [Rez97]. Their application potential may be restricted to modifications of existing library designs [Dew01]. They appear as by-products from code written for the design of a specific project [Lev98] or may be difficult to use [CFD]. Few have facilities for determining the MEMS manufacturing parameters as their primary purpose [MEM]. The availability of virtual reality in this area is very limited indeed.

To address the shortage of MEMS specific design tools, we have initiated ***M**EMS **A**nimated **G**raphic **D**esign **A**id* (MAGDA) for virtual prototyping, with a strong emphasis on visualizations. It embodies Computer-Aided Design (CAD) tools for modeling and simulating the functioning of MEMS in virtual reality and to provide visualizations of their behavior and performance as multi-parameter functions. It is intended to overcome some shortages in some of the large and popular CAD tools, by complementing, rather than replacing already existing MEMS software.

Its application niche is the exploration for determining the MEMS manufacturing dimensions and aids in confining them. The functioning of a mechanical device depends on its geometry and dimensions; consequently they have also an effect on the reliability through their design, choice of materials and wear out. We pay special attention to the effects of geometries on the functioning of the MEMS. With these effects in mind, we aim for a robust design.

In an innovative way, MAGDA combines visualizations to display both the geometries of the device as it is being designed, and animated functioning, an attribute that is in part affected by those geometries [Sit03, Li05]. One important feature that makes MAGDA different from other CAD software is that we use transparency in most of our visualizations, to be able to observe structures that are hidden otherwise. Our method is particularly suitable for the upcoming 3D Imaging and Holography displays

The paper is organized in the following way: Section 2 provides a brief overview of what makes MEMS different and their fabrication. This is followed by section 3 where we explain about etching and ways of modeling etching. In section 4 we explain our adaptation of the Marker/String method specifically for etching, and in section 5 we present our two experimental application cases and discussion. Finally section 6 brings the conclusions, with suggestions for future work.

2. MEMS BACKGROUND

MEMS are minute devices that are in widespread use, for example in airbag triggers and inkjet print heads, optical, medical, and many other applications. With ever increasing new applications in the R&D phase, the MEMS industry is strong and growing, in particular in the medical and optical applications. This in turn requires adequate development tools with sophisticated modeling and simulation software to reduce the lengthy prototyping and optimization period.

By their very nature MEMS devices are microscopic and therefore difficult to observe. In the macroscopic world of our daily experience inertia and gravity dominate the motion of objects. In contrast, in the microscopic domain of MEMS adhesion and friction are the dominant forces. Therefore MEMS designers cannot use their intuition on how things behave. Because of the different dominant forces, MEMS cannot simply be downscaled counterparts of larger mechanical machines, requiring innovative designs and arrangements of their components, whose effects are often not fully understood.

MEMS have emerged from the Integrated Circuit (IC) manufacture, which has revolutionized the world and started just a few decades ago. They are produced hundreds of thousands at one time on a Silicon wafer, a disc of silicon 5 to 30 cm in diameter, and less than a millimeter thick. In a sequence of alternating depositing layers of material, which are then specifically patterned (lithography) by removing parts of the material in specific patterns so that the desired structures emerge. Examples of the kinds of material that are deposited or grown in

layers are typically materials involving silicon or silicon oxides, but also metals. Due to the relatively recent MEMS industry, this often requires a lengthy and expensive cycle of trial and error. Silicon technology allows the construction of MEMS devices with a few micrometers (μm) in size, and whose structures are in the submicron range. In these processes etching is a fundamental processing step.

Another technique for producing a MEMS or parts of it is by producing a negative mould of the desired structure and the positive structure is then cast in metal or polymer (LIGA). The parts are then assembled into the micro system together with the regulating micro-circuitry. These devices are about two or more millimeters in size.

There are many more processing methods in a variety of sophistication and complexity, but for our purpose is not necessary to go deeper into the subject, for the interested reader a variety of introductory books are available, for example [Fat97], [Lys01].

3. ETCHING AND ITS MODELLING

Etching is the removal of material by mechanical or chemical methods. There are two types of etching isotropic and anisotropic. In isotropic etching the material removal occurs in all directions equally, while anisotropic etching occurs at different rates in different crystalline directions. It is fast in one direction, and slow, almost negligible or none in another direction, thus making it suitable for selective etching and straight walls.

Typically in anisotropic etching the crystalline orientation of the material to be etched is exploited and used in conjunction with masks and the technology for specific desired results, for example a V-grove for optical MEMS. Anisotropic etching is the preferred technique in integrated circuit manufacturing, where straight lines are common, as it can be controlled to very fine precision.

Dry etching is anisotropic, where the material removal is done by electrochemical or mechanical means such as Reactive Ion Etching, Plasma Etching, Sputtering, a.o. Wet etching is usually an isotropic process, where a mask is applied to obtain the desired shape, and with enough time, this mask can be underetched.

Isotropic etching cannot be so easily controlled, as the result depends much on the purity and age of all materials involved (including the masking material). However, in MEMS, with its variety of shapes and larger dimensions, isotropic wet etching is often preferred.

As a progressive material removal, etching can be modeled with Finite Element Analysis (FEA). FEA calculations are known to be slow and hence not very suitable for interactive VR visualizations. Simulations using the latest software package releases run on a common PC range from about 10 minutes for fast and simple calculations, to almost an hour for a moderately complex model. Much depends on the meshing and required refinements.

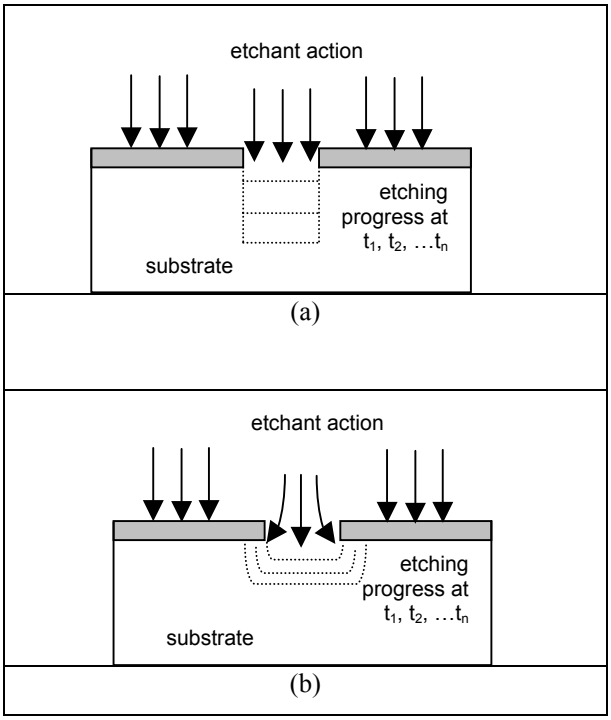


Figure 1 Different types of etching: (a) anisotropic etching progresses in one direction only; and (b) isotropic etching progresses in all directions equally.

Another problem is that sometimes the calculations of the given case do not converge. This is perhaps why etching simulation models have not been seen much in MEMS CAD. Coventor [Cov05] has only recently announced its etching module, but it is for anisotropic etching only. Anisotropic etching is bounded by flat planes, hence easier to model.

In isotropic etching however, the main difficulty is in determining a parametric model for the curved surfaces, as their curvature changes throughout the process. This is difficult because the shape of the set of surfaces (or curves if in 2D) has to be determined specifically for the materials involved (substrate and etchant) and for each and every shape of mask. A general equation cannot be derived for all the variety

of possible shapes of the structures in MEMS. An important work on etching has been prepared by Elwenspoek and Jansen [Elw98]. In general, MEMS structures require a variety and complexity of shapes, and the time and etching rate based on the chemical reactions are the sole options for controlling the etching progress. The usage of recipes for etching is common practice in clean rooms, hence, etching rates for different materials and etchants are well known and documented.

Underetching is the type of progress when portions of the substrate are etched out although they are masked. Underetching occurs from sideward etching excavating underneath the mask, as isotropic etching progresses. While underetching is normally undesired, it is important for MEMS in freeing mobile structures such as springs and gears. This has to be carefully controlled. Too much underetch and the structure to be freed is lost, too little underetch and the structure does not come free.

4. THE MARKER/STRING METHOD

In our endeavor to produce fast visualizations we need a model that is fast in calculation and capable of addressing complex structures, for example a set of combs, a spiral or any other shape. Amongst a range of approaches we have chosen to adapt the Marker/string method [Ada95] to our etching requirements. This method has been successfully applied in growth and solidifications in a slightly different way than for etching. One of the weaknesses in this method is that in sharp pointed structures (e.g. in the case of a complex mask shape), a swallow-tail condition can occur from the overlap. We have found a way to overcome this for the purpose of modeling etching. In this paper we focus on isotropic etching only, with its curved surfaces.

We can summarize our modeling approach applied for isotropic etching with the following points.

- *Surface.* The shape of any surface is characterized by vectors perpendicular to it. In the case of etching we have to find the vectors that are perpendicular to the surface (bowl) while it is being etched.
- *Grid.* We assume that the etchant is in contact with that surface. We set a grid and step size on the area that is not covered by the mask on the initially flat surface.
- *Direction.* We set a time step and initiate the process by selecting a point. We perform the cross product of the vectors that define the chosen point and its adjacent point on the grid. The resulting vector is perpendicular to the etched surface (with a small error which depends on the stepsize).
- *Progress.* We move to the next point in the grid, and then layer by layer. We repeat performing the cross products. The magnitude of the resulting vector is the vectorial sum of the vector at the current point, plus the new vector resulting from the cross product, with magnitude "etching rate".
- *Horizontal component.* In the moment when the etching profile passes below the lower edge of the mask, and in each new row, we introduce a horizontal vector of magnitude etching rate on either side. This is legitimate under the assumption that etching occurs equally in all directions, floor or wall. The horizontal vectors, together with the vertical vectors result in vectors at different angles, and give rise to the rounded corners of the etched bowl- shape. The side vector also initiates and governs the progress of the underetching process.

Validation

The application of this method in the way we are doing is valid, because at any time step we progress by an etching rate unit. At any one time the resulting vectors are perpendicular to the new surface profile.

To avoid swallow-tails on mask corners, we have performed a rotation. In this way we have preserved the etching rate. We can do this for isotropic etching, but it would not be valid for material deposition as proposed in [Ada95].

At this time we have treated all directions equally. We have not yet dealt with the crystalline orientation, which can affect slightly the etching rate in the case of isotropic etching, such that it is different in one direction from another. This can be resolved by applying the appropriate etching rate magnitude to the vectors in the model, that is, using a variety of etching rates. This would be the generalization of the method for both, iso-, and anisotropic etching.

5. EXPERIMENTS AND DISCUSSION

To test this method, we have applied the method to (a) a circular mask, and (b) to a square mask. Both cases require different treatment. For both cases we have calculated the wire mesh with the Marker/String method as adapted in the previous section, and then rendered the mesh. In both cases we are using Si and Silicon etchant in the proportion (126 HNO_3 : 60 H_2O : 5 NH_4F) for our simulations. We are simulating a 10 μm thick mask, with an opening with 45μm radius for the round mask; and 90×90 μm opening for the square mask.

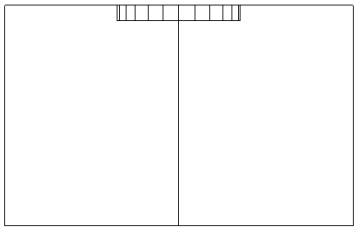
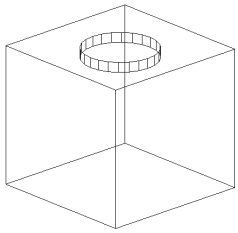
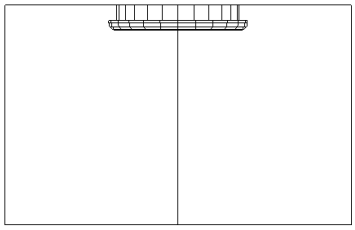
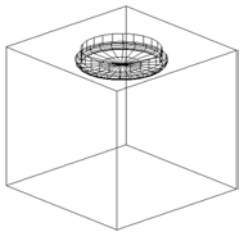
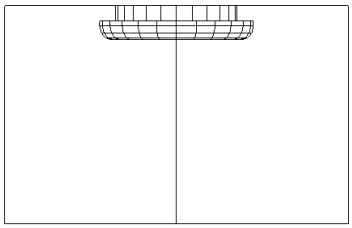
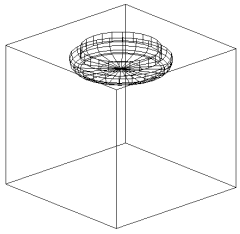
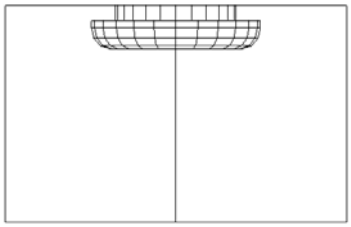
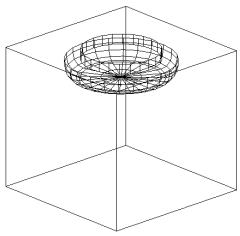
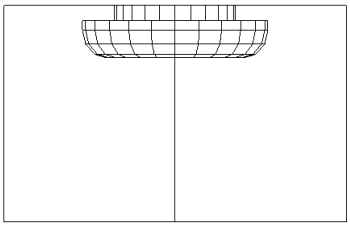
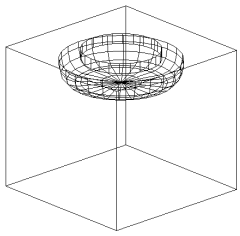
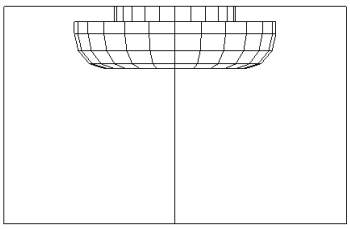
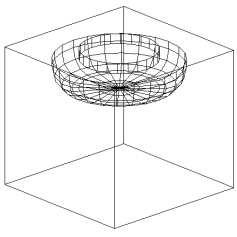
Round mask	*View: 2D*	*View: 3D*
Etch through mask time 0 -66.67 min		
Underetch at 106.67 min		
Underetch at 146.67 min		
Underetch at 186.67 min		
Underetch at 226.67 min		
Underetch at 266.67 min		

Figure 2 Progress of the etching surface at different time steps for the case of a 45 μm radius round mask opening

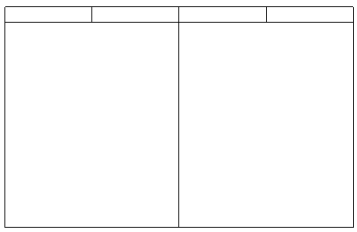
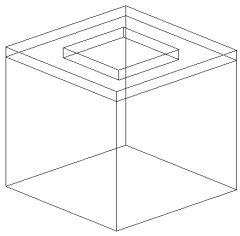
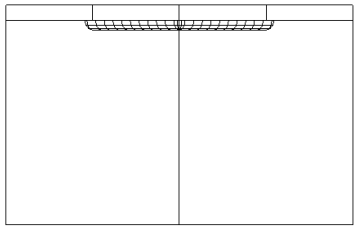
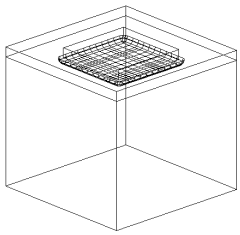
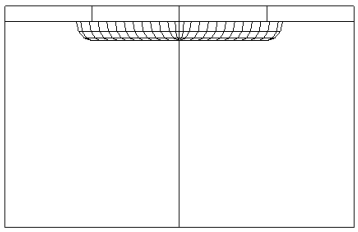
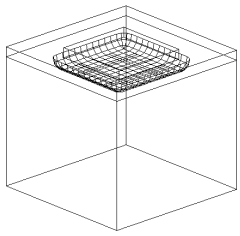
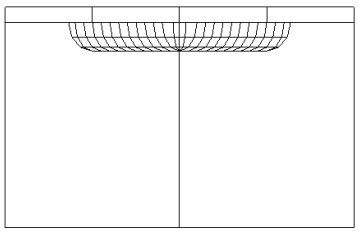
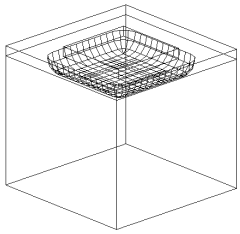
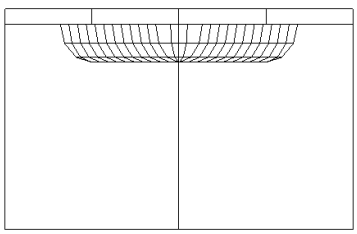
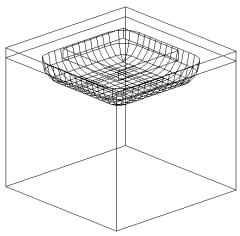
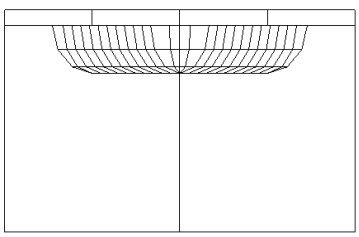
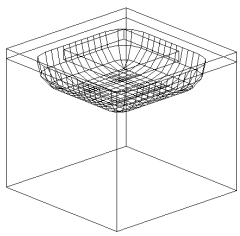
Round mask	*View: 2D*	*View: 3D*
Etch through mask time 0 -66.67 min		
Underetch at 106.67 min		
Underetch at 146.67 min		
Underetch at 186.67 min		
Underetch at 226.67 min		
Underetch at 266.67 min		

Figure 3 Progress of the etching surface at different time steps for the case of a 90×90 μm square mask opening

The calculations for the round mask are carried out in 2D. The resulting profile is then rotated stepping at an angle of 22.5°. The square mask is calculated in 3D for one quadrant, and then mirrored for the other quadrants. The progress of the etched surface is shown in the sequence of the wire frames in Figure 2 for the round mask, and Figure 3 for the square mask respectively. For the rendering we have chosen transparency, following our main philosophy in MAGDA. This is to allow better observation of structures that may be hidden by other structures, in this case the progress of the etched surface. Figure 4 shows the progress as rendered images at three different time steps.

In general we have found that our Marker/String adaptation works well for our purposes. The calculations are fast, results are obtained within seconds to simulate the progress of etching in a dynamic way for the wire-mesh. We have found that the underetched profile works satisfactory for both types of mask. However, for pointed corners a different etching rate must be used. This may slow down the process somewhat because at those corners the region for steeper etching has to follow a specific contour, merging later with the normal etching process, that is, changing continuously. Such a region is repetitively regular in the case of - for example - the pegs of a comb, but is different when irregular shapes are masked. High diversity together with high complexity in the shape of mask, will inevitably delay somewhat the calculation process.

The Marker/string method as we are using it, is affected by the error that is given by the assumption that distances between two points of the grid are straight, even when they are on a curved surface. This error is relatively small. It can be minimized with a smaller grid step at critical places such as corners and sharp edges. There is the inevitable trade off between accuracy and calculation time. However, for the purpose of visualizations this error is negligible and fades away in the visual representation by the resolution size of the pixels. Given that the method for calculating the wire-mesh is so fast, it is anticipated that this will not be a major problem. A systematic analysis of the optimal error minimization and grid size is envisaged for the future.

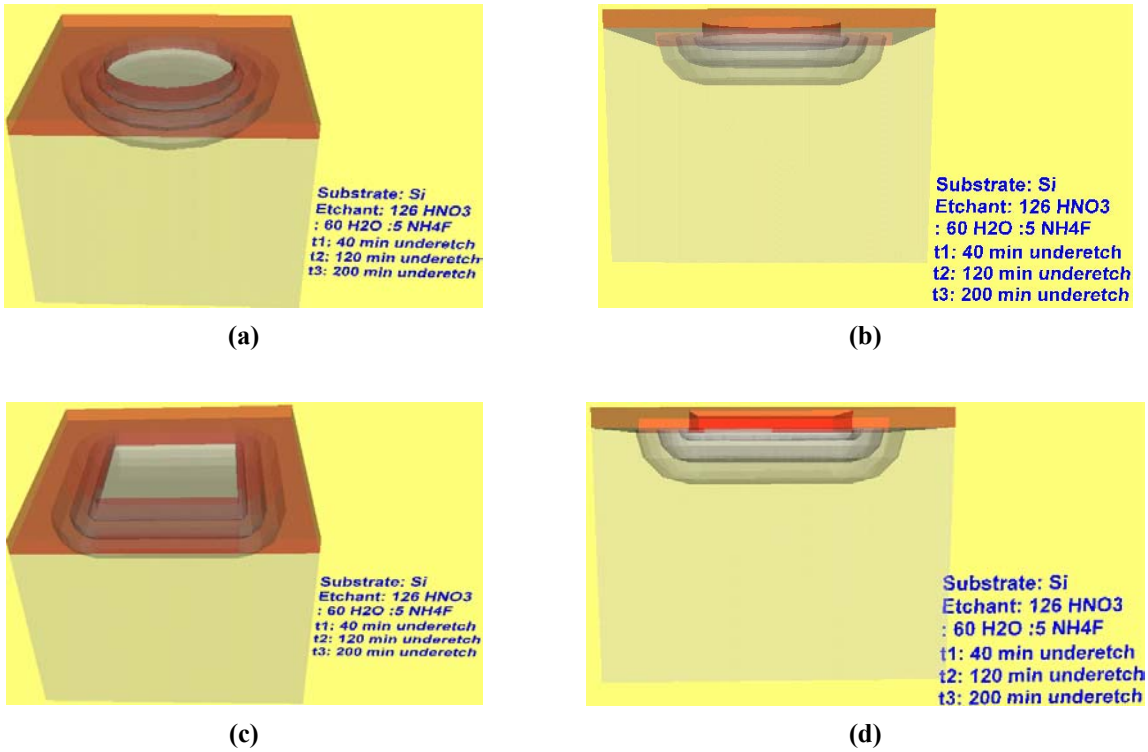


Figure 4 Rendered images of the etched surface with (a) round, and (b) square mask showing progress a three different time intervals

6. CONCLUSION

In this paper we have presented an adaptation of the Marker/String method to model isotropic etching, Our aim is to produce fast models that are suitable for interactive CAD for Micro Electro Mechanical Systems (MEMS). Our interest is for isotropic etching to cover the variety of shapes that appear in MEMS, but the method can be applied as well for anisotropic etching. We have demonstrated the model with two application cases, one for a round mask, and one for a square mask. In both cases, fast calculations and satisfactory results were obtained. Future work is aimed at using the level set approach for further speeding up the models and allow for more sophisticated structural complexity.

7. REFERENCES

[Ada95] D. Adalsteinsson, J.A. Sethian, "A Level Set Approach to a Unified Model for Etching, Deposition, and Lithography I: Algorithms and Two-Dimensional Simulations" Journal of Computational Physics Vol 120, 1995, pp 128-144

[ANS] ANSYS/Multiphysics, ANSYS Inc., Canonsburg, PA, 15317, USA, www.ansys.com.

[CFD] CFD-ACE+, CFD Research Corporation, www.cfdrc.com, 660 S. Bernardo Ave, Suite 4, Sunnyvale, CA, 94087, USA.

[Cov05] CoventorWare 2001.1, Coventor, Cary, NC, 27513, USA, www.coventor.com.

[Elw98] M. Elwenspoek, H.V. Jansen, "Silicon Micromachining", Cambridge University Press 1998

[Fem05] Femlab, Comsol, 744 Cowper Street Palo Alto, CA 94301, USA, www.comsol.com

[Fat97] S. Fatikow, U. Rembold, "Mycrosystems Technology and Microrobotics", Springer 1997

[Int05] IntelliSuite, IntelliSense, Cummings Park, Woburn MA 01801, USA, www.intellisense.com.

[Jhan02] A. S. Jhandi, R. Sitte, "Virtual Etching Aiding in MEMS Design", International Conference on Information Technology & Applications (ICITA'2002), Bathurst/Sydney, Australia (2002)

[Kar97] J. M. Karam, B. Courtois, H. Boutamine, P. Drake, A. Poppe, V. Szekely, M. Rencz, K. Hofmann, and M. Glesner, "CAD and Foundries for Microsystems", Proc. of the 34th Conference on Design Automation (DAC '97), Anaheim, CA, USA, pp. 674-679, 1997.

[Lev98] S.P.Levitan, T.P.Kurzweg, P.J.Marchand, M.A.Rempel, D.M.Chiarulli, J.A.Martinez, J.M. Bridgen, C.Fan, F.B McCormick, "Chatoyant, a Computer-Aided Design Tool for Free-Space Optoelectronic Systems", Applied Optics, Volume 37, Number 26, September 1998, pp. 6078-6092.

[Lys01] S.E. Lyschevski, "MEMS and NEMS Systems, Devices and Structures", CRC Press 2001

[Li01] Z. Li, R. Sitte, Zhaoyi Li, Renate Sitte, "Simulated Stroboscopic Illumination for Unsynchronized Motion Dynamics." Proceedings of the European Simulation and Modeling Conference ESMc_2003, Naples pp 299-303

[MEM] MEMS PRO, MEMSCAP, Oakland, CA, 94612, USA, www.memscap.com.

[Rez97] D. Reznik, S. Brown, J. Canny, "Dynamic Simulation as a Design Tool for a Microactuator Array", Proceedings IEEE Conference of Robotics and Automation (ICRA), Albuquerque, NM, April 1997, pp. 1675-1680.

[Sit03] R. Sitte, Visualizing Reliability in MEMS VR-CAD Tool Journal of WSCG, Vol 11, No 3 2003, pp 433-439