The 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2007

in co-operation with

EUROGRAPHICS

W S C G ' 2007

University of West Bohemia
Plzen
Czech Republic

January 29 – February 1, 2007

Full Papers Proceedings

Co-Chairs

Jarek Rossignac, Georgia Tech, USA
Vaclav Skala, University of West Bohemia, Plzen, Czech Republic

Edited by
Jarek Rossignac, Vaclav Skala

WSCG'2007 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 978-80-86943-98-5*

WSCG 2007

International Programme Committee

Alexa, Marc (Germany)
Bartz, Dirk (Germany)
Bekaert, Philippe (Belgium)
Benes, Bedrich (U.S.A.)
Bengtsson, Ewert (Sweden)
Bieri, Hanspeter (Switzerland)
Biri, Venceslas (France)
Bouatouch, Kadi (France)
Chen, Min (U.K.)
Chrysanthou, Yiorgos (Cyprus)
Coquillart, Sabine (France)
Davis, Larry (U.S.A.)
Deussen, Oliver (Germany)
du Buf, Hans (Portugal)
Ferguson, Stuart (U.K.)
Ferko, Andrej (Slovakia)
Goebel, Martin (Germany)
Groeller, Eduard (Austria)
Hauser, Helwig (Austria)
Hege, Hans-Christian (Germany)
Jansen, Frederik (The Netherlands)
Klosowski, James T. (U.S.A.)
Kobbelt, Leif (Germany)
Kruijff, Ernst (Austria)
Magnor, Marcus (Germany)
Midkiff, Sam (U.S.A.)
Myszkowski, Karol (Germany)
Pasko, Alexander (Japan)
Peroche, Bernard (France)
Puppo, Enrico (Italy)
Purgathofer, Werner (Austria)
Rauterberg, Matthias (The Netherlands)
Rokita, Przemyslaw (Poland)
Rossignac, Jarek (U.S.A.)
Rudomin, Isaac (Mexico)
Sbert, Mateu (Spain)
Schilling, Andreas (Germany)
Schumann, Heidrun (Germany)
Shamir, Ariel (Israel)
Skala, Vaclav (Czech Republic)
Sochor, Jiri (Czech Republic)
Teschner, Matthias (Germany)
Theoharis, Theoharis (Greece)
Veltkamp, Remco (The Netherlands)
Weiskopf, Daniel (Canada)
Wu, Shin-Ting (Brazil)
Wyvill, Brian (Canada)
Zemcik, Pavel (Czech Republic)

WSCG 2007 Board of Reviewers

Adzhiev,Valery (U.K.)
Alessandro, Piva (Italy)
Ammon, Lorenz (Switzerland)
Ancuti, Cosmin (Belgium)
Andreadis, Ioannis (Greece)
Andujar, Carlos (Spain)
Aspragathos, Nikos (Greece)
Aveneau, Lilian (France)
Bamidis, Panagiotis (Greece)
Bargouli, Maria (Greece)
Bartz, Dirk (Germany)
Battiato, Sebastiano (Italy)
Bekaert, Philippe (Belgium)
Bellon, Olga (Brazil)
Benes, Bedrich (USA)
Bengtsson, Ewert (Sweden)
Beyer, Johanna (Austria)
Bieri, Hanspeter (Switzerland)
Bilbao, Javier J. (Spain)
Biri, Venceslas (France)
Bottino, Andrea (Italy)
Bouatouch, Kadi (France)
Bourdin, Jean-Jacques (France)
Brunet, Pere (Spain)
Bruni, Vittoria (Italy)
Brunnet, Guido (Germany)
Buehler, Katja (Austria)
Butz, Andreas (Germany)
Callieri, Marco (Italy)
Clasen, Malte (Germany)
Coleman, Sonya (U.K.)
Coquillart, Sabine (France)
Crosnier, Andre (France)
Daniel, Marc (France)
Davis, Larry (USA)
de Geus, Klaus (Brazil)
Deussen, Oliver (Germany)
Dingliana, John (Ireland)
du Buf, Hans (Portugal)
Duce, David (U.K.)
Durikovic, Roman (Slovakia)
Egges, Arjan (The The Netherlands)
Eisemann, Martin (Germany)
Erbacher, Robert (USA)
Faudot, Dominique (France)
Feito, Francisco (Spain)
Felkel, Petr (Czech Republic)
Ferguson, Stuart (U.K.)
Ferko, Andrej (Slovakia)
Fernandes, Antonio (Portugal)
Fischer, Jan (Germany)
Flaquer, Juan (Spain)
Galin, Eric (France)
Galo, Mauricio (Brazil)
Ganovelli, Fabio (Italy)
Garcia-Alonso, Alejandro (Spain)
Gautron, Pascal (France)
Giannini, Franca (Italy)
Goebel, Martin (Germany)
Gonzalez, Pascual (Spain)
Grammalidis, Nicolaos (Greece)
Gresh, Donna (USA)
Groeller, Eduard (Austria)
Gutierrez, Diego (Spain)
Habbecke, Martin (Germany)
Haber, Tom (Belgium)
Hadwiger, Markus (Austria)
Harding, Bruce (USA)
Haro, Antonio (USA)
Hast, Anders (Sweden)
Hauser, Helwig (Austria)
Havran, Vlastimil (Czech Republic)
He, Xiangjian (Australia)
Hege, Hans-Christian (Germany)
Horain, Patrick (France)
Hornung, Alexander (Germany)
House, Donald (USA)
Chen, Min (U.K.)
Chin, Seongah (Korea)
Chover, Miguel (Spain)
Chrysanthou, Yiorgos (Cyprus)
Iglesias, Andres (Spain)
Ihrke, Ivo (Germany)
Jaillet, Fabrice (France)
Jansen, Frederik (The The Netherlands)
Joan-Arinyo, Robert (Spain)
Kähler, Ralf (Germany)
Kalantari, Leila (Canada)
Karabassi, Evaggelia-Aggeliki (Greece)
Kersten, Marta (Germany)
Kinuwaki, Shinichi (Germany)
Kjelldahl, Lars (Sweden)
Klajnsek, Gregor (Slovenia)
Klosowski, James T. (USA)
Knight, Michael (U.K.)
Kobbelt, Leif (Germany)
Kohlmann, Peter (Austria)
Kolcun, Alexej (Czech Republic)
Kruijff, Ernst (Austria)
Lamecker, Hans (Germany)
Lanquetin, Sandrine (France)
Larboulette, Caroline (Austria)
Lee, Seungyong (Korea)
Leitao, Miguel (Portugal)
Lensch, Heindrik (Germany)
Lewis, J.P. (USA)
Linz, Christian (Germany)
Magillo, Paola (Italy)
Magnor, Marcus (Germany)
Malik, Muddassir (Austria)
Mandl, Thomas (Germany)
Mantler, Stephan (Austria)
Matey, Luis (Spain)
Mattausch, Oliver (Austria)
McMenemy, Karen (U.K.)
Megali, Giuseppe (Italy)
Meneveaux, Daniel (France)
Moccozet, Laurent (Switzerland)
Mokhtari, Marielle (Canada)
Mollá Vayá, Ramón (Spain)
Montrucchio, Bartolomeo (Italy)
Mould, David (Canada)
Mudur, Sudhir (Canada)
Muller, Heinrich (Germany)
Muntean, Adrian (Germany)
Myszkowski, Karol (Germany)

Nedel, Luciana P. (Brazil)
Neveu, Marc (France)
Nielsen, Frank (Japan)
Ogayar, Carlos,J. (Spain)
Oliveira, Manuel (U.K.)
Pasko, Alexander (Japan)
Patel, Daniel (Austria)
Patow, Gustavo (Spain)
Paulin, Mathias (France)
Pavic, Darko (Germany)
Pedrini, Helio (Brazil)
Peroche, Bernard (France)
Pettifer, Steve (U.K.)
Platis, Nikos (Greece)
Plemenos, Dimitri (France)
Podorelec, David (Slovenia)
Ponchio, Federico (Italy)
Popov, Stefan (Germany)
Prakash, Edmond (U.K.)
Pratikakis, Ioannis (Greece)
Přikryl, Jan (Czech Republic)
Puppo, Enrico (Italy)
Purgathofer, Werner (Austria)
Rauterberg, Matthias (The The Netherlands)
Redon, Stephane (France)
Remolar, Inmaculada (Spain)
Renaud, Christophe (France)
Revelles, Jorge (Spain)
Ribelles, Jose (Spain)
Robert, Philippe (Switzerland)
Rodeiro, Javier (Spain)
Rodrigues, Marcos (U.K.)
Rojas-Sola, José Ignacio (Spain)
Rokita, Przemyslaw (Poland)
Rossignac, Jarek (USA)
Roth, Peter,M. (Austria)
Rudomin, Isaac (Mexico)
Sanna, Andrea (Italy)
Sbert, Mateu (Spain)
Scateni, Riccardo (Italy)
Segura, Rafael (Spain)
Sellent, Anita (Germany)
Semwal, Sudhanshu (USA)
Shamir, Ariel (Israel)
Scheiblauer, Claus (Austria)
Schilling, Andreas (Germany)
Schmidt, Thomas (Germany)
Schneider, Bengt-Olaf (USA)
Schumann, Heidrun (Germany)
Sips, Mike (USA)
Sirakov, Nikolay Metodiev (USA)
Sochor, Jiri (Czech Republic)
Solis, Ana Luisa (Mexico)
Sousa, A.Augusto (Portugal)
Stroud, Ian (Switzerland)
Suarez Rivero, Jose (Spain)
Suescun, Angel (Spain)
Szekely, Gabor (Switzerland)
Štulić, Radovan (Serbia and Montenegro)
Tang, Wen (U.K.)
Tawara, Takehiro (Japan)
Tecchia, Franco (Italy)
Teschner, Matthias (Germany)
Theußl, Thomas (Austria)
Tobler, Robert (Austria)
Tonet, Oliver (Italy)
Turini, Giuseppe (Italy)
Tytkowski, Krzysztof (Poland)
Vanecek, Petr (Czech Republic)
Vasa, Libor (Czech Republic)
Veiga, Luis (Portugal)
Veltkamp, Remco (The The Netherlands)
Vergeest, Joris (The The Netherlands)
Vitulano, Domenico (Italy)
Wan, Taoruan (U.K.)
Weigel, Christian (Germany)
Weiskopf, Daniel (Canada)
Wenger, Thomas (Switzerland)
Wu, Shin-Ting (Brazil)
Wyvill, Brian (Canada)
Zach, Christopher (Austria)
Zachmann, Gabriel (Germany)
Zalik, Borut (Slovenia)
Zemcik, Pavel (Czech Republic)
Zhu, Ying (USA)
Zimeras, Stelios (Greece)

Contents

Please, find the ”journal” papers in the Journal of WSCG

Paper Code		Title	Page
		Session A **Chair:Vergeest,J.S.M.**	
A43	A full	Alsweis,M.: Extended Competition Rules for Interacting Plants (Germany)	1
D67	A full	Chourasia,A., Schulze,J.: Data Centric High Dynamic Range Transfer Functions for Volume Rendering (United States)	9
F19	A full	Kuss,A., Hege,H.-C., Krofczik,S., Borner,J.: Pipeline for the Creation of Surface-based Averaged Brain Atlases (Germany)	17
G31	A journal	Lee,J.K., Newman,T.S.: New Method for Opacity Correction in Oversampled Volume Ray Casting (United States)	1
H31	A full	Desroche.S.,Jolivet.V.,Plemenos.D.: Towards Plan-based Automatic Exploration of Virtual Worlds (France)	25
		Session B **Chair: Alexa,M.**	
B03	B journal	Holst,M., Schumann,H.: Normal Mapping for Surfel-Based Rendering (Germany)	9
B67	B full	Turchyn,P.: Memory-efficient Sliding Window Progressive Meshes (Finland) Presentation	33
C97	B full	Tejada,E., Schafhitzel,T., Ertl,T.: Hardware-accelerated Rendering of Point-based Surfaces and Volumes (Germany)	44
G03	B full	Heymann,S., Müller,K., Smolic,A., Fröhlich,B., Wiegand,T.: SIFT Implementation and Optimization for General-Purpose GPU (Germany)	317
G19	B journal	Kim,S.-S., Nam,S.-W., Kim,D.-H., Lee,I.-H.: Hardware-Accelerated Ray-Triangle Intersection Testing for High-Performance Collision Detection (Korea)	17
		Session C **Chair: Smolic,A.**	
B13	C full	Tomaszewska,A., Mantiuk,R.: Image Registration for Multi-exposure High Dynamic Range Image Acquisition (Poland)	49
C23	C full	Vucini,E., Gokmen,M., Groller, E.: Face Recognition under Varying Illumination (Austria)	57
C79	C full	Ivanovska,T., Linsen,L.: Converting RGB Volume Data to Scalar Fields for Segmentation Purposes (Germany)	65
G23	C full	Steinmetz,S., Pellenz,J., Paulus,D.: Estimation of Planar Surfaces in Noisy Range Images for the RoboCup Rescue Competition (Germany)	73
D73	C full	Schwartz,W.R., Pedrini,H.: Color Textured Image Segmentation Based on Spatial Dependence Using 3D Co-occurrence Matrices and Markov Random Fields (Brazil)	81

Session D **Chair: Kolingerova,I.**

D11	D journal	Hutter,M., Fuhrmann,A.: Optimized Continuous Collision Detection for Deformable Triangle Meshes (Germany)	25
E83	D full	Juckel,T., Beckhaus,S.: Rendering Diffuse Objects using Particle Systems inside Voxelized Surface Geometry (Germany)	89
F07	D journal	Spillmann,J., Becker,J.M., Teschner,M.: Non-iterative Computation of Contact Forces for Deformable Objects (Germany)	33
F17	D full	Carvalho,S.R., Boulic,R.,Thalmann,D.: Motion Pattern Preserving IK Operating in the Motion Principal Coefficients Space (Switzerland)	97
G97	D full	De Martino,J.M., Violaro,F.: Benchmarking Speech Synchronized Facial Animation Based on Context-Dependent Visemes (Brazil)	105

Session E **Chair:Bottino,A.**

B59	E full	Sanjurjo,J.R., Amor,M., Bóo,M., Doallo,R.: Improving Locality for Progressive Radiosity Algorithm: A Study based on the Blocking Transformation of the Scene. (Spain)	113
B71	E journal	Vivanloc,V., Hoelt,J.Ch., Hong,C.-B., Paulin,M.: Light Octree: Global Illumination Fast Reconstruction and Realtime Navigation (France)	41
C89	E journal	Linsen,L., Mueller,K., Rosenthal,P.: Splat-based Ray Tracing of Point Clouds (Germany)	51
E79	E full	Umenhoffer,T., Szirmay-Kalos,L.: Robust Diffuse Final Gathering on the GPU (Hungary)	121
G13	E full	Cadik,M.: Perception Motivated Hybrid Approach to Tone Mapping (Czech Republic)	129

Session F **Chair: Vanecek,P.**

D41	F full	Wang,H., Vergeest,J.S.M, Song,Y., Wiegers,T.: Automated 3D Scan Multi-view Registration Based on Rotation Estimation (Netherlands)	137
E97	F full	Langerak,T.R., Vergeest,J.S.M., Wang,H., Song,Y.: An Evolutionary Strategy for Free Form Feature Identification in CAD Models (Netherlands)	145
F31	F full	Domonkos,B., Csébfalvi,B.: Interactive Distributed Translucent Volume Rendering (Hungary)	153
H53	F full	Musialski,P., Tobler,R.F., Maierhofer,S.: Smooth Subdivision Surfaces over Multiple Meshes (Germany)	161
E43	F full	Gupta,R.B., Lee,B.-G., Lee,J.-J.: A new Image Interpolation Technique Using Exponential B-spline (Korea)	169

Session G **Chair: Kohout,J.**

A89	G full	Seoane,A., Taibo,J., Hernández,L., López,R., Jaspe,A.: Hardware-Independent Clipmapping (Spain)	177
C11	G full	Dimitrov,D., Knauer,Ch., Kriegel,K., Rote,G.: Upper and Lower Bounds on the Quality of the PCA Bounding Boxes (Germany)	185
G11	G full	Fairén,M., Trueba,R.: Octree-based View-dependent Triangle Meshes (Spain)	193
C43	G full	Livny,Y., Kogan,Z., El-Sana,J.: Seamless Patches for GPU-Based Terrain Rendering (Israel)	201
G07	G full	Pina,A., Bentes,C., Farias,R.: Memory Efficient and Robust Software Implementations of the Raycast Algorithm (Brazil)	209

Session H **Chair: Kolingerova,I.**

B17	H journal	Scheider,M., Klein,R.: Efficient and Accurate Rendering of Vector Data on Virtual Landscapes (Germany)	59
B29	H journal	Bade,R., Konrad,O., Preim,B.: Reducing Artifacts in Surface Meshes Extracted from Binary Volumes (Germany)	67
C71	H journal	Shi,K., Theisel,H., Weinkauf,T., Hauser, H., Hege,H.-C., Seidel,H.-P.: Extracting Separation Surfaces of Path Line Oriented Topology in Periodic 2D Time-Dependent Vector Fields (Germany)	75
E37	H journal	Kohlmann,P.,Bruckner,S.,Kanitsar,A.,Gröller,E.: Evaluation of a Bricked Volume Layout for a Medical Workstation based on Java (Austria)	83
E61	H journal	Schafhitzel,T., Falk,M., Ertl,T.: Real-Time Rendering of Planets with Atmospheres (Germany)	91

Session K **Chair: Vasa,L.**

C31	K journal	Amjoun,R.: Efficient Compression of 3D Dynamic Mesh Sequences (Germany)	99
C67	K full	Rudnick,S., Linsen,L., McPherson,E.G.: Inverse Modeling and Animation of Growing Single Stemmed Trees (Germany)	217
D03	K full	Knight,J.K., Semwal,S.K.: Fast Skeleton Estimation from Motion Captured Data using Generalized Delogne-Kasa method (United States)	225
E05	K full	Nakajima,H., Sugisaki,E., Morishima,S.: Tweakable Shadows for Cartoon Animation (Japan)	233
E23	K full	Iglesias,R., Prada,E., Uribe,A., Garcia-Alonso,A., Casado,S., Gutierez,T.: Assembly simulation on collaborative haptic virtual environments (Spain)	241

Session L **Chair: Rossignac,J.**

A19	L full	Castelló,P., Sbert,M., Chover,M., Feixas,M.: Viewpoint Entropy-Driven Simplification (Spain)	249
A73	L full	Porcu,M., Scateni,R.: Dimensional Induced Clustering for Mesh Recognition (Italy)	257
F53	L journal	Medek,P., Benes,P., Sochor,J.: Computation of Tunnels in Protein Molecules using Delaunay Triangulation (Czech Republic)	107
F71	L full	Trenkel,S., Weller,R., Zachmann,G.: A Benchmarking Suite for Static Collision Detection Algorithms (Germany)	265
G29	L full	Levet,F., Granier,X., Schlick,Ch.: Triangulation of Uniform Particle Systems: Its Application to the Texturation of Implicit Surfaces (France)	271
H07	L full	Sun,S.,Huang,D.W.: Efficient Region-Based Pencil Drawing (China)	279

Session M **Chair: Zemcik,P.**

A41	M full	Darles,E., Crespin,B., Ghazanfarpour,D.: Accelerating and Enhancing Rendering of Realistic Ocean Scenes (France)	287
A53	M full	Kuehl,B., Blom,K., Beckhaus,S.: Generation of Shadows in Scene Graphs based VR (Germany)	295
B07	M journal	Götzelmann,T., Götze,M., Ali,K., Hartmann,K., Strothotte,Th.: Annotating Images through Adaptation: An Integrated Text Authoring and Illustration Framework (Germany)	115
G67	M journal	Habel,R., Wimmer,M., Jeschke,S.: Instant Animated Grass (Austria)	123
A83	M full	Hasic,J., Chalmers,A.: Visual Attention for Significantly Influencing the Perception of Virtual (Bosnia and Herzegovina)	303
A31	M full	Li,Y., Lizhuang,M., Di,W.: Fast Colorization Using Edge and Gradient Constrains (China)	309

Session N **Chair: Hast,A.**

F47	N journal	Lourakis,M., Argyros,A.: Refining Single View Calibration With the Aid of Metric Scene Properties (Greece)	129
G59	N journal	Bottino,A., De Simone,M., Laurentini,A.: Recognizing Human Motion using Eigensequences (Italy)	135
H61	N journal	Ancuti,C., Bekaert,P.: A Scale Invariant Detector Based on Local Energy Model for Matching Images (Belgium)	143
A47	N journal	Chu,Ch., Shih,Z.-Ch.: Painterly Rendering Framework from Composition (Taiwan)	151

FOREWORD

This book contains the proceedings of WSCG'2007, the 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2007. The Winter School of Computer Graphics started in 1992 with about 20 participants from the former Czechoslovakia. It has become an important conference attended by the international Computer Graphics community.

Submissions to WSCG'2007 came from numerous countries on different continents. From these submissions, 186 papers have been sent to reviewers. Each paper was reviewed by at least three experts selected among the members of the large International Program Committee. 61 papers have been accepted to be presented at WSCG'07, representing a 33% acceptance ratio. The best 20 of these papers have been selected to be published in a special issue of the Journal of WSCG and will be available through the conference Digital Library at **http://wscg.zcu.cz/DL/wscg_DL.htm**.

Furthermore, the published papers are listed on the major international scientific indexes such as Thomson ISI.

WSCG'2007 features three key-note lectures:

- Shape Comparison, Simplification, and Compression by J. Rossignac from Georgia Tech, USA
- Intuitive Editing of Surface Meshes by M. Alexa from T.U. Berlin, Germany
- 3D Video and Free Viewpoint Video – Technologies, Applications and MPEG Standards, by A. Smolic, from the Heinrich-Hertz-Institut, Germany

Finally, we would like to express our thanks to all the authors who submitted technical papers to WSCG'2007, and to the reviewers whose work and dedication made it possible to put together a very exciting program of high technical quality.

The Program Co-Chairs

Jarek Rossignac, Georgia Institute of Technology, Atlanta, USA

Vaclav Skala, University of West Bohemia, Plzen, Czech Republic, Plzen

Hardware-Independent Clipmapping

Antonio Seoane
antonio.seoane@videalab.udc.es

Javier Taibo
jtaibo@udc.es

Luis Hernández
lhernandez@udc.es

Rubén López
ryu@videalab.udc.es

Alberto Jaspe
jaspe@udc.es

VideaLAB
School of Civil Engineering - University of A Coruña, Spain
Campus de Elviña 15071 - Spain

ABSTRACT

We present a technique for efficient management of large textures and its real-time application to geometric models. The proposed technique is inspired by the *clipmap* [12] idea, that caches in video memory a subset of the texture mipmap pyramid. Based on this concept, we define some structures and a different management allowing its implementation on a personal computer without specific graphics hardware. Finally, we present the results of the application in a terrain visualization system, using several simultaneous textures with a detail up to 0.25 meters per texel, covering a 60,000 km^2 area.

Keywords
Terrain visualization, multiresolution visualization, large data set visualization, level-of-detail techniques, texture mapping, clipmap, mipmap, texture caching.

1 INTRODUCTION

Applying textures to digital terrain models is the classical solution to simulate the missing geometry details. When we want to apply a large amount of texture to the terrain surface, the storage problem arises. Although both the system memory and the video memory are extremely fast, there is limited storage capacity. Therefore the use of a paging technique that allows an efficient management of the texture data in order to visualize the terrain with a very high quality is essential.

In contrast to the abundance of terrain geometry management algorithms, there is little work focused on handling the texture. Moreover, most of the systems allowing the visualization of digital terrain models establish a strong dependency between both geometry and texture data bases.

Usually, texture tiles are bound to the geometry with a pre-established mapping. This makes it difficult to modify or to replace the geometry or texture data in a transparent way, without rebuilding the database.

One of the best and most used approaches that allows the handling of big textures is the *clipmap* [12]. This technique separates the handling of the texture and the geometry, allowing independence between both databases. The main problem of this technique is the requirement of specific hardware.

We propose a new technique that allows the handling of a large amount of texture without any requirement of specific hardware. Only OpenGL or Direct3D fixed function pipeline is required to implement this technique. It uses a two level cache composed of a texture stack stored in the video memory as the first level and a set of buffers stored in RAM as the second level. The contents of both levels are updated depending on camera movement.

This technique is inspired by the clipmap idea. It is also based on the caching in video memory of a huge mipmap pyramid [13]. Nevertheless, its structure, the management of the video memory and the way the texture is applied are different, which allows its implementation in a personal computer using a graphics API like OpenGL [11].

Our texturing technique provides the following advantages:

- It can be implemented using an API like OpenGL without special necessities in the graphics hardware.
- It keeps the independence between geometry and texture databases.

 ISBN 978-80-86943-98-5

- Texture coordinates can be computed in the GPU (although it is not necessary), avoiding their transference to the graphics system. This allows modification of the geometry in real time, while keeping the right texture mapping without recomputing the texture coordinates.
- Texture aliasing is avoided using trilinear filtering hardware capabilities.
- It allows the visualization of high resolution textures with the possibility of including higher resolution insets.
- It allows the use of several independent large textures which can be combined to show different information types simultaneously on the terrain.

2 PREVIOUS WORK

Historically, the strategies for the terrain texturing problem have been based on paging systems. With these kind of techniques it's usual to solve the high texture volume problem applying high resolution information to the terrain regions closer to the camera, and less detail to those which are further away. Thus we only need to store in video memory the higher resolution data for a small portion of the terrain. Moreover, due to the perspective, the further we are from a terrain area, the fewer pixels used on the screen, and therefore, the fewer texels needed for accurate representation.

Rabinovich [9] proposed in his visualization system the use of a single texture covering the whole terrain, with hardware mipmap. This technique has scalability drawbacks, as the maximum texture size allowed by graphics hardware is very limited, and greatly exceeded by the resolution needed to represent large terrain surfaces with an acceptable visual quality.

A solution to the previous limitation is the clipmap [12], which caches a subset of a mipmap pyramid. Terrain is mapped at every point with the finest available level of detail. As the camera moves, the system updates the pyramid cache with the information corresponding to the new area. The clipmap, nevertheless, needs special hardware for its implementation.

Another solution is MPGrid [6], that uses several pyramids instead of only one. Each pyramid is a mipmap that should fit completely in memory. If the geometry is not aligned with the texture pyramids, it is necessary to either clip the triangles in real-time or to use a special hardware.

Döllner [5] proposes to store a tree containing a set of texture patches that belong to one texture pyramid. Each texture patch is bound to a geometry patch of another multiresolution model, which must cover it completely. This introduces a dependency that forces the adaptation of texture quality to the loaded geometry level and vice versa.

Blow [2] developed a similar system, in which the tree is a quadtree. For every triangle, the more adequate texture is sought in the quadtree. To achieve efficiency in this algorithm in spite of doing the above mentioned computation once for every triangle, he introduces certain restrictions in the way of doing the clipping and the shape of the triangles. These restrictions create, also in this case, a strong dependency between the geometry and texture systems.

3 STRUCTURE OVERVIEW

The system proposed in this paper is based in the clipmap concept described by Tanner, caching a subset of the texture pyramid in video memory (Figure 1). We introduce a different memory structure of the algorithm, described in this section.

Texture information is structured in three storage levels: disk, system memory (RAM) and video memory (VRAM).

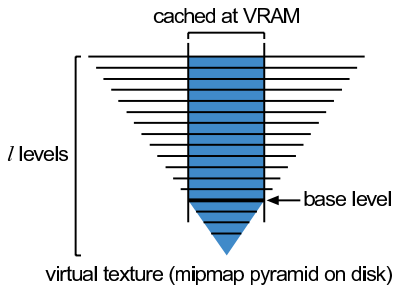


Figure 1: Virtual texture.

3.1 Disk

Texture is completely stored on disk using a mipmap pyramidal scheme. This texture is called *virtual texture*. The highest detail level of this pyramid is formed by $2^{l-1} \times 2^{l-1}$ texels, where l is the number of levels (Figure 1). Levels are numbered from 0 and up, being $2^i \times 2^i$ the size for level i. Higher levels can be incomplete, allowing incrementation of detail for special interest areas (*insets*).

Every level is stored on disk being divided in square fragments of $t \times t$ texels, called *tiles*, except those levels which dimension is smaller than the tile. Tile size is chosen so as to maximize the speed of disk to RAM transfer, while avoiding fragmentation. Tiles are addressed using three coordinates: x (column), y (row) and z (level).

The disk database is fully compatible with the OpenGL Performer [10] clipmap format.

The amount of disk space used by the texture is then estimated with the equation:

$$D \simeq 2^{2l-2} \cdot \frac{4}{3} \cdot b \text{ bytes}$$

where b is the texture color depth measured in bytes per texel.

 ISBN 978-80-86943-98-5

3.2 System memory

Disk stored tiles are cached in a set of RAM buffers, one buffer per tile.

Cache requests address the tiles by column, row and level. Tile loading is done asychronously, and an LRU algorithm is used to choose the buffer where the tile is to be stored.

Requests are prioritized on a level basis, the coarser levels being those with higher priority. This facilitates having information of the zone of interest as quickly as possible, since lower level tiles cover larger areas. The detail is then being progressively and orderly refined as higher level tiles become available.

RAM usage is determined by the number of buffers (n) assigned to the cache, being calculated as follows:

$$R = n \cdot t^2 \cdot b \text{ bytes}$$

3.3 Video memory

A subset of the virtual texture pyramid stored in disk is stored in video memory (Figure 1).

Clip size (c) determines the maximum size stored in VRAM for each level of the virtual texture. We will choose a *base level*, this being the level of the pyramid with a size equal to the clip size. This base level and lower ones are stored entirely and permanently in VRAM. For each level higher than the base level, a region of $c \times c$ texels around the center of detail is cached. The base level (l_b) is calculated as follows:

$$l_b = \log_2 c$$

Information placed in VRAM is organized in a graphics system texture stack, being those textures independent of each other. The stack is composed by $l - l_b$ textures of $c \times c$ texels each. Texture t_i caches level $l_b + i$ in the pyramid (Figure 2).

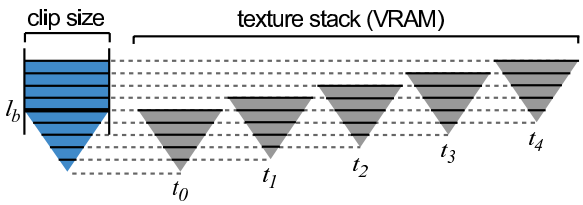


Figure 2: Texture stack.

In order to allow the graphic system to perform a trilinear filtering to avoid aliasing, mipmap levels for every texture are needed. Let t_{ij} be the mipmap level j of the texture i in the stack (Figure 3).

Texture t_0 has all its mipmap levels, corresponding with the base level and coarser levels. For the rest of the textures, level t_{ij} caches level $l_b + i - j$ of the virtual texture. In these textures, the number of mipmap levels can be limited to save bandwidth.

We can visualize this stack as a set of rings representing different resolutions (Figure 4). In the Figure 5 we

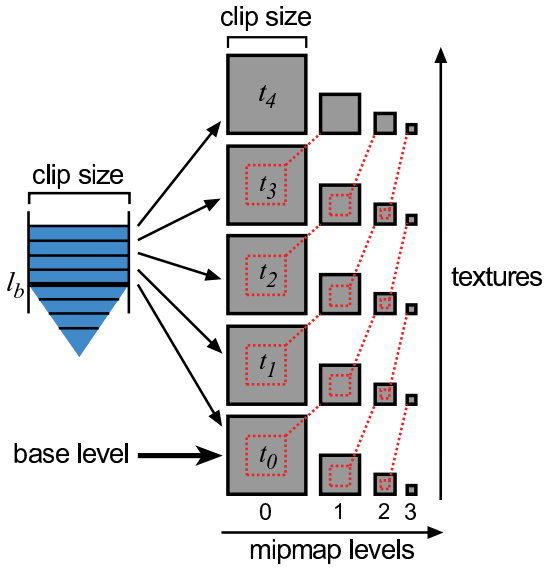


Figure 3: Texture stack. Mipmap levels correspondence.

can see the application of different levels of detail to the terrain, represented by a color code.

Figure 4: Rings of detail.

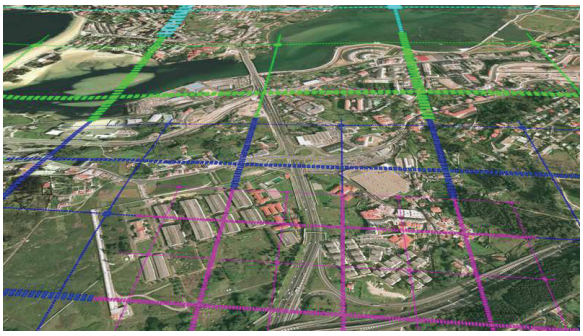


Figure 5: Virtual textures applied to the terrain. Levels of detail are shown using a color code.

To compute the usage of video memory let us supose that m mipmap levels are used for the textures corre-

 ISBN 978-80-86943-98-5

sponding to those levels of the disk pyramid above the base level. It is computed as

$$V = \left((l - l_b - 1) \cdot \sum_{i=0}^{m} \left(\frac{c}{2^i} \right)^2 + \sum_{i=0}^{l_b+1} 2^{2i} \right) \cdot b$$

4 UPDATE

The data stored on the texture stack, as described in the previous section, corresponds to a zone of the virtual texture around the center of detail. As the center of detail position is moved, it is necessary to update the contents of the stack and other related structures.

Center of detail. For every frame, the application must place the center of detail in the location where higher quality is desired. Several strategies can be used, usually computing it as a function of the camera position and orientation.

The simplest approach is to place the center of detail in the vertical projection of the camera location over the ground. More adequate approaches place the center of detail on a point of the visible geometry close to the camera.

Texture stack update. Each texture of the VRAM texture stack is updated independently, caching a level of the virtual texture. VRAM textures are considered to be divided in square patches, called *subtiles*. The subtile is the texture updating unit. The subtile size (s) must be a divisor of the clip size and the tile size, with

$$s = 2^i,\ t = 2^j,\ c = 2^k,\ \text{with } i \leq j \text{ e } i < k$$

After the center of detail is moved, some subtiles will retain useful data, but other tiles must be updated. Each texture has a state matrix that keeps the state of its subtiles. Subtiles that must be loaded with new data are marked in the state matrix as invalid.

Processing the textures of the stack in ascending order, each invalid subtile is loaded from the tile that contains the information it needs at that moment. If the requested tile is in the cache in RAM, the subtile is updated. Otherwise, the subtile remains invalid, waiting for the needed tile to be loaded from disk. In case of incomplete levels, subtiles in the areas where there is no information will never be updated.

To keep the coherence of data, the described subtile update implies updating the related area in every mipmap level of the texture. Update of levels t_{ij}, where $j > 0$, can be made from lower textures in the stack because this data is replicated in several textures (Figures 2 and 3). In this way data can be transferred inside VRAM, faster than loading it from RAM.

There is a toroidal structure of the subtiles in VRAM due to efficiency reasons. Considering the virtual texture levels divided in subtiles, subtile (x_a, y_a) from level i on disk is placed in position (x_b, y_b) inside the texture in VRAM, where

$$(x_b, y_b) = (x_a \bmod K, y_a \bmod K) \text{ and } K = \frac{2^i}{s}$$

Load control. The speed of the center of detail affects the amount of texture that must be updated. The higher detail textures cover a smaller area than the lesser detail ones and they must be updated more frequently. The time required to update the texture stack in VRAM can cause the time available for rendering the frame to be overrun. To avoid this potential problem, it is necessary to restrict the amount of time available for updating the texture stack.

The stack update is made from the coarser texture in ascending order. A texture from the stack must be completely updated before the update of the next texture begins. No matter how quickly the center of detail is moving, there will always be a set of textures fully updated (at least the base level).

Computing the subtile size, it is important to find a tradeoff between an adequate load control and a good transfer rate. The less the subtile size is, the higher the accuracy to measure the update time will be. Even though the subtile update time is strongly dependent on hardware used, usually the smaller sizes have a very poor efficiency.

Concentric rings updating. As the textures in the stack are not always completely updated, it is necessary to decide when a texture is appliable. A simple approach is to discard a texture from the stack until it is completely updated. The problem here is that every time the center of detail is moved the distance of a subtile, it will be invalidated until being completely updated again. This problem is reduced by applying the texture when a partial area of the full texture area is loaded. This subarea is called texture *coverage*.

We update the subtiles in concentric rings, innermost to outermost, so the coverage grows as the subtile rings are updated (Figure 6). This way the texture is useful from the moment it begins to have valid subtiles. Beginning from the center, the highest interest zone is available sooner. Also, the center subtiles are the ones with higher life expectancy.

5 RENDERING

To use the described technique in a real application, we have to do the following tasks

- Apply the texture
- Compute the texture coordinates
- Draw the geometry

There are two possible approaches to doing these tasks:

 ISBN 978-80-86943-98-5

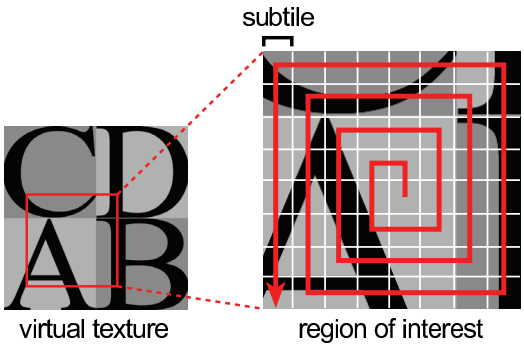


Figure 6: Circular update.

1. For each geometry set, apply the finest available texture covering this region.

2. Apply each texture from the stack, asking for its coverage and drawing the geometry covered by this level but not for the finer ones.

For texture mapping the geometry, we need the virtual texture coordinates bound to each vertex. Texture coordinate computation is made in exactly the same way, no matter which one of the two approaches we choose.

Virtual texture coordinates are converted to coordinates of the texture selected from the stack. Texture coordinates are scaled because each texture from the stack covers half the virtual space of the previous one. Thus, the scale factor for the level i from the stack is computed as $S = 2^i$. We only need the texture matrix of the fixed function pipeline to automatically scale the texture coordinates. Once the coordinates are scaled, the toroidal organization assures that repeated application of the texture [3] will be correctly mapped over the covered surface (Figure 7).

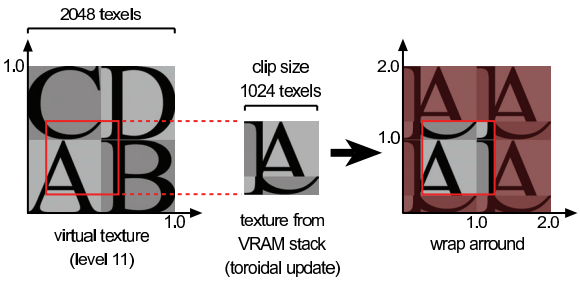


Figure 7: Mapping a texture from the stack.

5.1 Automatic generation of texture coordinates.

In terrain visualization, the usual way of mapping the geometry is to have the texture coordinates precalculated.

The computation of virtual texture coordinates can be done by the texturing system. The only information the texturing system needs to know about the geometry database is the coordinate system used. There can even be different projection systems for texture and geometry.

The real-time computation of these texture coordinates gives us some advantages, one of them being the ability to dinamically modify the geometry while keeping the mapping right.

The programmable capabilities of new graphics pipelines allows us to move these computations from CPU to GPU. This way, we achieve a significant reduction of bandwidth consumption between RAM and VRAM, and avoid the need of VRAM space to store texture coordinates. We can generate the virtual texture coordinates and scale them using a vertex program.

5.2 Multitexturing

An interesting feature of our technique is that several virtual textures can be managed simultaneously. Each one is bound to a texture stage in the graphics system. The maximum number of virtual textures is determined by the graphics hardware used.

In terrain visualization, we can check different types of information over the ground by mapping several overlapping textures (Figure 8).

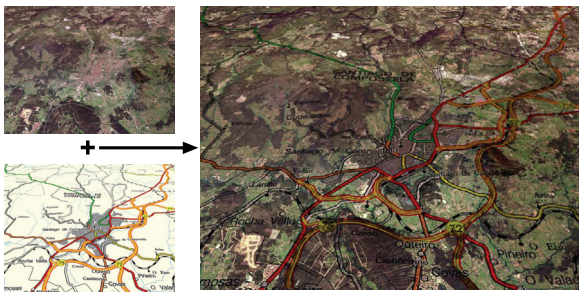


Figure 8: Virtual textures combination.

6 EXAMPLE OF APPLICATION: SANTI

An OpenGL [11] implementation of the technique described above has been applied in the last version of SANTI, a terrain visualization system [8] (Figure 9). SANTI is an application that was developed to display a very large area of Spain as part of a permanent exhibition from 1999 on. It was initially implemented on an SGI Infinite Reality architecture using clipmapping for terrain texture management. Currently it runs on a common personal computer.

This application combines several virtual textures simultaneously, using a vertex program to dynamically compute the texture coordinates.

The system uses an De Boer's [4] inspired algorithm to render the geometry dividing the total mesh into equally sized patches that can be drawn using different levels of detail. There is no constraint on the way the patches are generated. The texture level of detail

 ISBN 978-80-86943-98-5

is calculated on a block basis, and it has no relation at all with the geometry level of detail. Distant patches are considered as a block in themselves, while close patches are split considering every new subpatch as a block for texturing purposes. Using this approach, the geometry granularity is increased near the camera, to display higher textural detail.

Figure 9: Real-time terrain visualization (SANTI).

This system has been successfully used to visualize real cases of digital terrain models covering more than 60,000 km^2. The most detailed texture used to map this surface has a virtual size of $2^{20} \times 2^{20}$ texels, composing a pyramid with 21 levels of detail. In this pyramid, levels 0 to 16 are generated for the full geographical area, allowing full coverage of the terrain with a 5 meters per texel detail, while for special interest areas levels 17 to 20 are also generated to reach a texture detail of up to 0.25 meters per texel (Figure 10).

Figure 10: View of a 0.25 meters per texel area.

Texure is fragmented in tiles, each of them of 512×512 texels. Clip size measures 2048×2048 texels and the subtile size is 128×128 texels.

The application maintains a steady 75 fps display rate over all the terrain using a personal computer with a regular hardware configuration.

At present, authors are preparing a new 5 terabytes database with 21 full levels of detail that will allow their users to visualize the above mentioned 60,000 Km^2 terrain with a continuous 0.25 meters per texel resolution.

7 RESULTS AND DISCUSSION

The system has been tested with a real data set. These tests were done in a low end personal computer with the following features: Intel Pentium 4 2.8 GHz with 512 MB DDR RAM, GeForce 4 Ti4800SE with 128 MB, AGP 8x, SATA disk 7200 rpm with an approximate bandwidth of 53 MB/s.

The rendered data set includes two combined virtual textures of Galicia (northwest of Spain), simultaneously showing satellite image and a road map (Figure 8) for a 250×250 km area. The resolution of the satellite image is 5 meters per texel over all the area, with one aerial image inset of 0.5 meters per texel, covering a 32×32 km area and other of 0.25 meters per texel, covering a 2.5×2.5 km (21 levels). The resolution of the road map image is 16 meters per texel (15 levels). These images are mapped over a regular 200×200 m cell sized terrain mesh.

Both virtual textures have a clip size of 1024 texels, a subtile size of 128 texels, a color depth of 24 bits per texel. There is a maximum allowed update time of 1 ms per frame for each virtual texture.

The test was a flight over the insets with 0.5 and 0.25 resolution. The speed of flight was 250 meters/s.

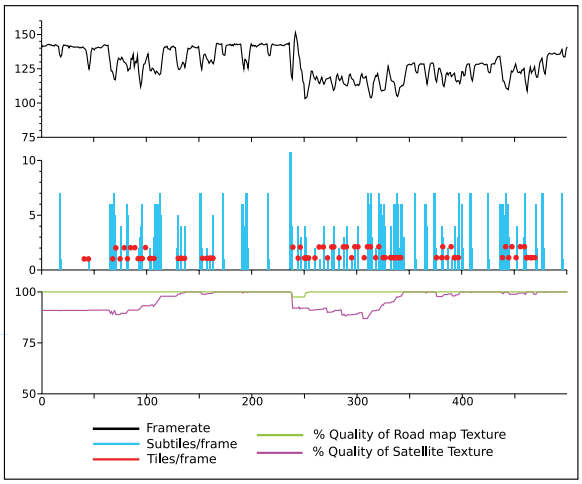


Figure 11: Test results.

The Figure 11 shows a graph for a 4 seconds interval. There we can compare the frame rate with the number of subtile and tile updates, as well as the quality available for both textures. The quality of a texture is measured as the percentage of updating of the texture stack.

As seen in Table1, the average quality of the virtual textures is never under 95%. The update process has little influence on the frame rate, keeping it always over 75 fps.

The update of the texture stack uses an average bandwidth of 7.25 MB/s, approximately 55.29 KB/frame, while the update of the RAM cache needs a transfer rate of 17.4 MB/s, being 32.84% of disk total bandwidth.

 ISBN 978-80-86943-98-5

Globals	Min.	Max.	Avg.	Std. dev.
Frame rate (frames/s)	91.65	152.05	128.54	11.66
Subtile load (subtiles/frame)	0	14	1.17	2.32
Tile load (tiles/frame)	0	2	0.20	0.51
Tile load latency (ms)	0.96	126.15	15.83	17.62
Terrain texture				
Quality (%)	86.93	100.00	96.00	4.35
Subtile load (subtiles/frame)	0	7	1.13	2.19
Tile load (tiles/sec)	0	2	0.19	0.50
Road map texture				
Quality (%)	97.50	100.00	99.93	0.39
Subtile load (subtiles/frame)	0	7	0.05	2.32
Tile load (tiles/sec)	0	1	0.01	0.51

Table 1: Test statistics.

These results prove that our technique allows the management of multiple virtual textures with exceptional performance.

8 SUMMARY AND FUTURE WORK

We introduce a new technique for managing very large textures through a paging system. It keeps the texture and geometry databases independent of each other and it can be implemented on personal computers using standard graphic API's, like OpenGL, without the need of special hardware.

This technique has been applied on a real case displaying a 60,000 Km^2 terrain texture with a detail of up to 0.25 meters per texel with a steady 75 fps frame rate.

The dynamic, procedural generation of the texture is a research line that is currently open. This could be used for rendering vector data such as GIS information over a three-dimensional geometry, without the need of keeping the texture stored on disk.

Another line of research is to try different alternatives for real time texture decompression [7] [1] in order to reduce the bandwidth needed to transmit those textures through a network for LAN or Internet applications.

REFERENCES

[1] María José Abasolo and Francisco J. Perales López. Wavelet analysis for a new multiresolution model for large-scale textured terrains. *WSCG 2003*, 2003.

[2] J. Blow. Terrain rendering at high levels of detail. In *Proceedings of the Game Developers Conference*, 2000.

[3] Tom McReynolds David Blythe, Brad Grantham and Scott Nelson. Advanced graphics programming techniques using opengl. ACM SIGGRAPH 1998 Course #17 Notes, July 1998.

[4] Willem H. de Boer. Fast terrain rendering using geometrical mipmapping. 2000. Available in http://www.flipcode.com/tutorials/geomipmaps.pdf.

[5] Jürgen Döllner, Konstantin Baumman, and Klaus Hinrichs. Texturing techniques for terrain visualization. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 227–234. IEEE Computer Society Press, 2000.

[6] T. Hüttner. High resolution textures. *Visualization'98 - Late Breaking Hot Topics Papers*, pages 13–17, November 1998.

[7] Young-Su Kwon, In-Cheol Park, and Chong-Min Kyung. Pyramid texture compression and decompression using interpolative vector quantization. In *ICIP*, 2000.

[8] L.Hernández, J.Taibo, and A.Seoane. Una aplicación para la navegación en tiempo real sobre grandes modelos topográficos. In *IX Congreso Español de Informática Gráfica, CEIG 1999.*, 1999.

[9] Boris Rabinovich and Craig Gotsman. Visualization of large terrains in resource-limited computing environments. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 95–102. IEEE Computer Society Press, 1997.

[10] John Rohlf and James Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394. ACM Press, 1994.

[11] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.2.1)*. Editor (v1.1): Chris Frazier, Editor (v1.2): Jon Leech, March 1998. http://www.opengl.org/.

[12] Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones. The clipmap: a virtual mipmap. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 151–158. ACM Press, 1998.

[13] Lance Williams. Pyramidal parametrics. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11. ACM Press, 1983.

 ISBN 978-80-86943-98-5

Upper and Lower Bounds on the Quality of the PCA Bounding Boxes

Darko Dimitrov, Christian Knauer, Klaus Kriegel, Günter Rote
Freie Universität Berlin
Institute of Computer Science
Takustrasse 9, D-14195 Berlin, Germany

darko/knauer/kriegel/rote@inf.fu-berlin.de

ABSTRACT

Principle component analysis (PCA) is commonly used to compute a bounding box of a point set in $\mathbb{R}^d$. The popularity of this heuristic lies in its speed, easy implementation and in the fact that usually, PCA bounding boxes quite well approximate the minimum-volume bounding boxes. In this paper we give a lower bound on the approximation factor of PCA bounding boxes of convex polytopes in arbitrary dimension, and an upper bound on the approximation factor of PCA bounding boxes of convex polygons in $\mathbb{R}^2$.

Keywords

Bounding Boxes, Principal Component Analysis, Computational Geometry.

1. INTRODUCTION

Substituting sets of points or complex geometric shapes with their bounding boxes is motivated by many applications. For example, in computer graphics, it is used to maintain hierarchical data structures for fast rendering of a scene or for collision detection. Additional applications include those in shape analysis and shape simplification, or in statistics, for storing and performing range-search queries on a large database of samples.

Computing a minimum-area bounding box of a set of n points in $\mathbb{R}^2$ can be done in $O(n \log n)$ time, for example with the rotating caliper algorithm [Tou83]. O'Rourke [O'R85] presented a deterministic algorithm, a rotating caliper variant in $\mathbb{R}^3$, for computing the exact minimum-volume bounding box of a set of n points in $\mathbb{R}^3$. His algorithm requires $O(n^3)$ time and $O(n)$ space. Barequet and Har-Peled [BHP99] have contributed two $(1+\varepsilon)$-approximation algorithms for computing the minimum-volume bounding box for point sets in $\mathbb{R}^3$, both with nearly linear complexity. The running times of their algorithms are $O(n + 1/\varepsilon^{4.5})$ and $O(n \log n + n/\varepsilon^3)$, respectively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

Numerous heuristics have been proposed for computing a box which encloses a given set of points. The simplest heuristic is naturally to compute the axis-aligned bounding box of the point set. Two-dimensional variants of this heuristic include the well-known *R-tree*, the *packed R-tree* [RL85], the R^**-tree* [BKSS90], the R^+*-tree* [SRF87], etc.

A frequently used heuristic for computing a bounding box of a set of points is based on *principal component analysis*. The principal components of the point set define the axes of the bounding box. Once the axis directions are given, the dimension of the bounding box is easily found by the extreme values of the projection of the points on the corresponding axis. Two distinguished applications of this heuristic are the OBB-tree [GLM96] and the BOXTREE [BCG+96], hierarchical bounding box structures, which support efficient collision detection and ray tracing. Computing a bounding box of a set of points in $\mathbb{R}^2$ and $\mathbb{R}^3$ by PCA is quite fast, it requires linear time. To avoid the influence of the distribution of the point set on the directions of the PCs, a possible approach is to consider the convex hull, or the boundary of the convex hull $CH(P)$ of the point set P. Thus, the complexity of the algorithm increases to $O(n \log n)$. The popularity of this heuristic, besides its speed, lies in its easy implementation and

 ISBN 978-80-86943-98-5

in the fact that usually PCA bounding boxes are tight-fitting (see [LKM+00] for some experimental results).

Given a point set $P \subseteq \mathbb{R}^d$ we denote by $BB_{pca}(P)$ the PCA bounding box of P and by $BB_{opt}(P)$ the bounding box of P with smallest possible volume. The ratio of the two volumes $\lambda_d(P) = Vol(BB_{pca}(P))/Vol(BB_{opt}(P))$ defines the approximation factor for P, and

$$\lambda_d = \sup\left\{\lambda_d(P) \mid P \subseteq \mathbb{R}^d, Vol(CH(P)) > 0\right\}$$

defines the general PCA approximation factor. We are not aware of any previous published results about this quality feature of PCA. Here, we give lower bounds on λ_d for arbitrary dimension d, and an upper bound on λ_2.

The paper is organized as follows. In Section 2. we review the basics of principal component analysis. In particular, we present the continuous version of PCA, which results in the introduction of a series of approximation factors $\lambda_{d,i}$, where i ranges from 0 to d and denotes the dimension of the faces of the convex hull that contribute to the continuous point set for which the principal components are computed. In Section 3. we give lower bounds on $\lambda_{d,i}$ for arbitrary values of d and $1 \leq i \leq d$. An upper bound on $\lambda_{2,1}$ is presented in Section 4. We conclude with future work and open problems in Section 5.

2. PRINCIPAL COMPONENT ANALYSIS

The central idea and motivation of PCA [Jol02] (also known as the Karhunen-Loeve transform, or the Hotelling transform) is to reduce the dimensionality of a point set by identifying *the most significant directions (principal components)*. Let $X = \{x_1, x_2, \ldots, x_m\}$, where x_i is a d-dimensional vector, and $c = (c_1, c_2, \ldots, c_d) \in \mathbb{R}^d$ be the center of gravity of X. For $1 \leq k \leq d$, we use x_{ik} to denote the k-th coordinate of the vector x_i. Given two vectors u and v, we use $\langle u, v\rangle$ to denote their inner product. For any unit vector $v \in \mathbb{R}^d$, the *variance of X in direction v* is

$$var(X, v) = \frac{1}{m}\sum_{i=1}^{m}\langle x_i - c, v\rangle^2. \tag{1}$$

The most significant direction corresponds to the unit vector v_1 such that $var(X, v_1)$ is maximum. In general, after identifying the j most significant directions $B_j = \{v_1, v_2, \ldots, v_j\}$, the $(j+1)$-th most significant direction corresponds to the unit vector v_{j+1} such that $var(X, v_{j+1})$ is maximum among all unit vectors perpendicular to $v_1, v_2, \ldots, v_j$.

It can be verified that for any unit vector $v \in \mathbb{R}^d$,

$$var(X, v) = \langle Cv, v\rangle, \tag{2}$$

where C is the *covariance matrix* of X. C is a symmetric $d \times d$ matrix where the (i, j)-th component, $c_{ij}, 1 \leq i, j \leq d$, is defined as

$$c_{ij} = \frac{1}{m}\sum_{k=1}^{m}(x_{ik} - c_i)(x_{jk} - c_j). \tag{3}$$

The procedure of finding the most significant directions, in the sense mentioned above, can be formulated as an eigenvalue problem. If $\lambda_1 > \lambda_2 > \cdots > \lambda_d$ are the eigenvalues of C, then the unit eigenvector v_j for λ_j is the j-th most significant direction. All λ_js are non-negative and $\lambda_j = var(X, v_j)$. Since the matrix C is symmetric positive definite, its eigenvectors are orthogonal. If the eigenvalues are not distinct, the eigenvectors are not unique. In this case, an orthogonal basis of eigenvectors is chosen arbitrary. However, we can achieve distinct eigenvalues by a slight perturbation of the point set.

The following result summarizes the above background knowledge on PCA. For any set S of orthogonal unit vectors in $\mathbb{R}^d$, we use $var(X, S)$ to denote $\sum_{v \in S} var(X, v)$.

Lemma 1 *For $1 \leq j \leq d$, let λ_j be the j-th largest eigenvalue of C and let v_j denote the unit eigenvector for λ_j. Let $B_j = \{v_1, v_2, \ldots, v_j\}$, $sp(B_j)$ be the linear subspace spanned by B_j, and $sp(B_j)^\perp$ be the orthogonal complement of $sp(B_j)$. Then $\lambda_1 = \max\{var(X, v) : v \in \mathbb{R}^d, \|v\| = 1\}$, and for any $2 \leq j \leq d$,*

i) $\lambda_j = \max\{var(X, v) : v \in sp(B_{j-1})^\perp, \|v\| = 1\}$.

ii) $\lambda_j = \min\{var(X, v) : v \in sp(B_j), \|v\| = 1\}$.

iii) $var(X, B_j) \geq var(X, S)$ *for any set S of j orthogonal unit vectors.*

Since bounding boxes of a point set P (with respect to any orthogonal coordinate system) depend only on the convex hull of $CH(P)$, the construction of the covariance matrix should be based only on $CH(P)$ and not on the distribution of the points inside. Using the vertices, i.e., the 0-dimensional faces of $CH(P)$ to define the covariance matrix C we obtain a bounding box $BB_{pca(d,0)}(P)$. We denote by $\lambda_{d,0}(P)$ the approximation factor for the given point set P and by

$$\lambda_{d,0} = \sup\left\{\lambda_{d,0}(P) \mid P \subseteq \mathbb{R}^d, Vol(CH(P)) > 0\right\}$$

the approximation factor in general. The example in Figure 1 shows that $\lambda_{2,0}(P)$ can be arbitrarily large if

 ISBN 978-80-86943-98-5

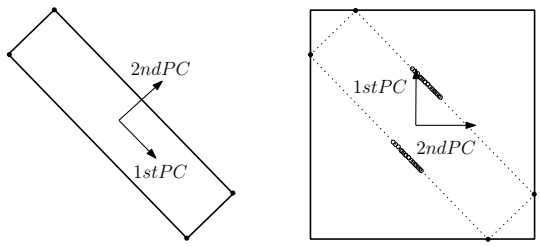


Figure 1: Four points and its PCA bounding-box (left). Dense collection of additional points significantly affect the orientation of the PCA bounding-box (right).

the convex hull is nearly a thin rectangle, but with a lot of additional vertices in the middle of the two long sides. Since this construction can be lifted into higher dimensions we obtain a first general lower bound.

Proposition 2 $\lambda_{d,0} = \infty$ *for any* $d \geq 2$.

To overcome this problem, one can apply a continuous version of PCA taking into account (the dense set of) all points on the boundary of $CH(P)$, or even all points in $CH(P)$. In this approach X is a continuous set of d-dimensional vectors and the coefficients of the covariance matrix are defined by integrals instead of finite sums.

Note that for for $d = 1$ the above problem is trivial, because the PCA bounding box is always optimal, i.e., $\lambda_{1,0}$ and $\lambda_{1,1}$ are 1.

2.1 Continuous PCA

Variants of the continuous PCA, applied on triangulated surfaces of 3D objects, were presented by Gottschalk et. al. [GLM96], Lahanas et. al. [LKM+00] and Vranić et. al. [VSR01]. In what follows, we briefly review the basics of the continuous PCA in a general setting.

Let X be a continuous set of d-dimensional vectors with constant density. Then, the center of gravity of X is

$$c = \frac{\int_{x \in X} x dx}{\int_{x \in X} dx}. \tag{4}$$

Here, $\int dx$ denotes either a line integral, an area integral, or a volume integral in higher dimensions. For any unit vector $v \in \mathbb{R}^d$, the *variance of X in direction v* is

$$var(X, v) = \frac{\int_{x \in X} \langle x - c, v \rangle^2 dx}{\int_{x \in X} dx}. \tag{5}$$

The covariance matrix of X has the form

$$C = \frac{\int_{x \in X} (x - c)(x - c)^T dx}{\int_{x \in X} dx}, \tag{6}$$

with its (i, j)-th component

$$c_{ij} = \frac{\int_{x \in X} (x_i - c_i)(x_j - c_j) dx}{\int_{x \in X} dx}, \tag{7}$$

where x_i and x_j are the i-th and j-th component of the vector x, and c_i and c_j i-th and j-th component of the center of gravity. It can be verified that relation (2) is also true when X is a continuous set of vectors. The procedure of finding the most significant directions, can be also reformulated as an eigenvalue problem and consequently Lemma 1 holds.

For point sets P in $\mathbb{R}^2$ we are especially interested in the cases when X represents the boundary of $CH(P)$, or all points in $CH(P)$. Since the first case corresponds to the 1-dimensional faces of $CH(P)$ and the second case to the only 2-dimensional face of $CH(P)$, the generalization to a dimension $d > 2$ leads to a series of $d - 1$ continuous PCA versions. For a point set $P \in \mathbb{R}^d$, $C(P, i)$ denotes the covariance matrix defined by the points on the i-dimensional faces of $CH(P)$, and $BB_{pca(d,i)}(P)$, denotes the corresponding bounding box. The approximation factors $\lambda_{d,i}(P)$ and $\lambda_{d,i}$ are defined as

$$\lambda_{d,i}(P) = \frac{Vol(BB_{pca(d,i)}(P))}{Vol(BB_{opt}(P))}, \quad \text{and}$$
$$\lambda_{d,i} = \sup\left\{\lambda_{d,i}(P) \mid P \subseteq \mathbb{R}^d, Vol(CH(P)) > 0\right\}.$$

3. LOWER BOUNDS

We start with straightforward conclusion from Proposition 2.

Proposition 3 $\lambda_{d,i} = \infty$ *for any* $d \geq 4$ *and any* $1 \leq i < d - 1$.

Proof. We can use a lifting argument to establish $\lambda_{k,i} \leq \lambda_{k+1,i+1}$, and thus $\lambda_{d,i} \geq \lambda_{d-1,i-1} \geq \ldots \geq \lambda_{d-i,0} = \infty$. □

This way, there remain only two interesting cases for a given d: the factor $\lambda_{d,d-1}$ corresponding to the boundary of the convex hull, and the factor $\lambda_{d,d}$ corresponding to the full convex hull. The nontrivial lower bounds we are going to derive are based on the following connection between the symmetry of a point set and its principal components.

Lemma 4 *Let P be a d-dimensional point set symmetric with respect to a hyperplane H and assume that the*

 ISBN 978-80-86943-98-5

covariance matrix C has d different eigenvalues. Then, a principal component of P is orthogonal to H.

Proof. Without loss of generality, we can assume that the hyperplane of symmetry is spanned by the last $d-1$ standard base vectors of the d-dimensional space and the center of gravity of the point set coincides with the origin of the d-dimensional space, i.e., $c = (0,0,\ldots,0)$. Then, the components c_{1j} and c_{j1}, for $2 \leq j \leq d$, are 0, and the covariance matrix has the form:

$$C = \begin{bmatrix} c_{11} & 0 & \ldots & 0 \\ 0 & c_{22} & \ldots & c_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & c_{d2} & \ldots & c_{dd} \end{bmatrix} \tag{8}$$

Its characteristic polynomial is

$$det(C - \lambda\, I) = (c_{11} - \lambda) f(\lambda), \tag{9}$$

where $f(\lambda)$ is a polynomial of degree $d-1$, with coefficients determined by the elements of the $(d-1) \times (d-1)$ submatrix of C. From this it follows that c_{11} is a solution of the characteristic equation, i.e., it is an eigenvalue of C and the vector (1, 0, ...,0) is its corresponding eigenvector (principal component), which is orthogonal to the assumed hyperplane of symmetry. □

3.1 Lower bounds in $\mathbb{R}^2$

The result obtained in this subsection can be seen as special case of the result obtained in the subsection 3.3. To gain a better understanding of the problem and the obtained results, we consider it separately.

Theorem 5 $\lambda_{2,1} \geq 2$ *and* $\lambda_{2,2} \geq 2$.

Proof. Both lower bounds can be derived from a rhombus. Let the side length of the rhombus be 1. Since the rhombus is symmetric, its PCs coincide with its diagonals. On the right side in Figure 2 its optimal-area bounding boxes, for 2 different angles, $\alpha > 90°$ and $\beta = 90°$, are shown, and on the left side its corresponding PCA bounding boxes. As the rhombus' angles in limit approach 90°, the rhombus approaches a square with side length 1, i.e., the vertices of the rhombus in the limit are $(\frac{1}{\sqrt{2}},0), (-\frac{1}{\sqrt{2}},0), (0,\frac{1}{\sqrt{2}})$ and $(0,-\frac{1}{\sqrt{2}})$ (see the left side in Figure 2), and the dimensions of its PCA bounding box are $\sqrt{2} \times \sqrt{2}$. According to Lemma 4, the PCs of the rhombus are unique

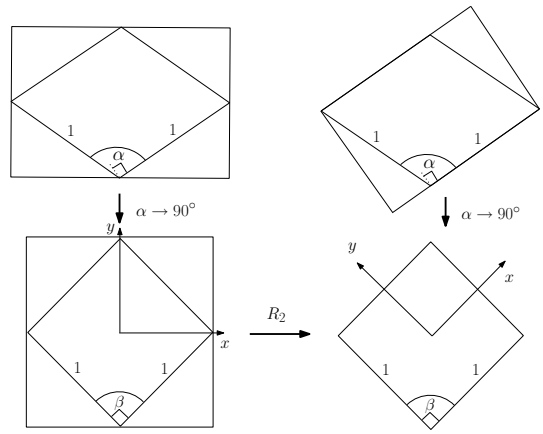


Figure 2: An example which gives us the lower bound of the area of the PCA bounding box of an arbitrary convex polygon in $\mathbb{R}^2$.

as long its angles are not 90°. This leads to the conclusion that the ratio between the area of the bounding box on the left side in Figure 3, and the area of its PCA bounding box, on the right side in Figure 3, in limit goes to 2. □

Alternatively, to show that the given squared rhombus fits into a unit cube, one can apply the following rotation matrix

$$R_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{10}$$

It can be verified easily that all coordinates of the vertices of the rhombus transformed by R_2 are in the interval $[-0.5, 0.5]$. We use similar arguments when we prove the lower bounds in higher dimensions.

3.2 Lower bounds in $\mathbb{R}^3$

Theorem 6 $\lambda_{3,2} \geq 4$ *and* $\lambda_{3,3} \geq 4$.

Proof. Both lower bounds are obtained from a dipyramid, having a rhombus with side length $\sqrt{2}$ as its base. The other sides of the dipyramid have length $\frac{\sqrt{3}}{2}$. Similarly as in $\mathbb{R}^2$, we consider the case when its base, the rhombus, in limit approaches the square, i.e., the vertices of the square dipyramid are $(1,0,0), (-1,0,0), (0,1,0), (0,-1,0), (0,0,\frac{\sqrt{2}}{2})$ and $(0,0,-\frac{\sqrt{2}}{2})$ (see the left side in Figure 3). The dimensions of its PCA bounding box are $2 \times 2 \times \sqrt{2}$. Now, we rotate the coordinate system (or the square

 ISBN 978-80-86943-98-5

dimension	$\mathbb{R}$	$\mathbb{R}^2$	$\mathbb{R}^3$	$\mathbb{R}^4$	$\mathbb{R}^5$	$\mathbb{R}^6$	$\mathbb{R}^7$	$\mathbb{R}^8$	$\mathbb{R}^9$	$\mathbb{R}^{10}$
lower bound	1	2	4	16	16	32	64	4096	4096	8192

Table 1: Lower bounds for the approximation factor of PCA bounding boxes for the first 10 dimensions.

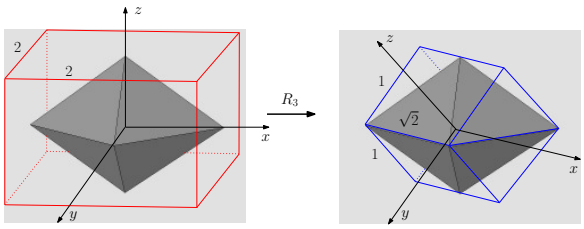


Figure 3: An example which gives the lower bound of the volume of the PCA bounding box of an arbitrary convex polygon in $\mathbb{R}^3$.

dipyramid) with the rotation determined by the following orthogonal matrix

$$R_3 = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{\sqrt{2}} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{\sqrt{2}} \end{bmatrix}. \tag{11}$$

It can be verified easily that the square dipyramid, after rotation with R_3 fits into the box $[-0.5, 0.5]^3$ (see the right side in Figure 3). Thus, the ratio of the volume of the bounding box, on the left side in Figure 3, and the volume of its PCA bounding box, on the right side in Figure 3, in limit goes to 4. □

3.3 Lower bounds in $\mathbb{R}^d$

Theorem 7 *If d is a power of two, then $\lambda_{d,d-1} \geq \sqrt{d}^d$ and $\lambda_{d,d} \geq \sqrt{d}^d$.*

Proof. For any $d = 2^k$, let a_i be a d-dimensional vector, with $a_{ii} = \frac{\sqrt{d}}{2}$ and $a_{ij} = 0$ for $i \neq j$, and let $b_i = -a_i$. We construct a d-dimensional convex polytope P_d with vertices $V = \{a_i, b_i | 1 \leq i \leq d\}$. It is easy to check that the hyperplane normal to a_i is a hyperplane of reflective symmetry, and as consequence of Lemma 4, a_i is an eigenvector of the covariance matrix of P_d. To ensure that all eigenvalues are different (which implies that the PCA bounding box is unique), we add $\varepsilon_i > 0$ to the i-th coordinate of a_i, and $-\varepsilon_i$ to the i-th coordinate of b_i, for $1 \leq i \leq d$, where $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_d$. When all ε_i, $1 \leq i \leq d$, arbitrary approach 0, the PCA bounding box of the convex polytope P_d converges to a hypercube with side lengths $\sqrt{d}$, i.e., the volume of the PCA bounding box of P_d converges to $\sqrt{d}^d$. Now, we rotate P_d, such that it fits into the cube $[-\frac{1}{2}, \frac{1}{2}]^d$. For $d = 2^k$, we can use a rotation matrix derived from a *Hadamard matrix*[1], recursively defined by

$$R_d = \frac{1}{\sqrt{2}} \left[\begin{array}{c|c} R_{\frac{d}{2}} & R_{\frac{d}{2}} \\ \hline R_{\frac{d}{2}} & -R_{\frac{d}{2}} \end{array} \right], \tag{12}$$

where we start with the matrix R_2 (10) defined above for $d = 2$. A straightforward calculation verifies that P_d rotated with R_d fits into the cube $[-0.5, 0.5]^d$. □

Remark: Theorem 7 holds for all dimensions d for which a $d \times d$ Hadamard matrix exists. Hadamard conjectured that this is the case for all multiples of four. This conjecture is known to be true for $d \leq 664$ [KTR05].

We can combine lower bounds from lower dimensions to get lower bounds in higher dimensions by taking Cartesian products. If λ_{d_1} is a lower bound for the ratio between the PCA bounding box and the optimal bounding box of a convex polytope in $\mathbb{R}^{d_1}$, and λ_{d_2} is a lower bound in $\mathbb{R}^{d_2}$, then $\lambda_{d_1} \cdot \lambda_{d_2}$ is a lower bound in $\mathbb{R}^{d_1+d_2}$. This observation together with the results from this section enables us to obtain lower bounds in any dimension. For example, for the first 10 dimensions, the lower bounds we obtain are given in Table 1.

4. AN UPPER BOUND FOR $\lambda_{2,1}$

Given a point set $P \subseteq \mathbb{R}^2$ and an arbitrary bounding box $BB(P)$ we will denote the two side lengths by a and b, where $a \geq b$. We are interested in the side lengths $a_{opt}(P) \geq b_{opt}(P)$ and $a_{pca}(P) \geq b_{pca}(P)$ of $BB_{opt}(P)$ and $BB_{pca(2,1)}(P)$, see Figure 4. The parameters $\alpha = \alpha(P) = a_{pca}(P)/a_{opt}(P)$ and $\beta = \beta(P) = b_{pca}(P)/b_{opt}(P)$ denote the ratios between the corresponding side lengths. Hence, we have $\lambda_{2,1}(P) = \alpha(P) \cdot \beta(P)$. If the relation to P is clear, we will omit the reference to P in the notations introduced above.

Since the side lengths of any bounding box are bounded by the diameter of P, we can observe that in

[1] A Hadamard matrix is a ± 1 matrix with orthogonal columns.

 ISBN 978-80-86943-98-5

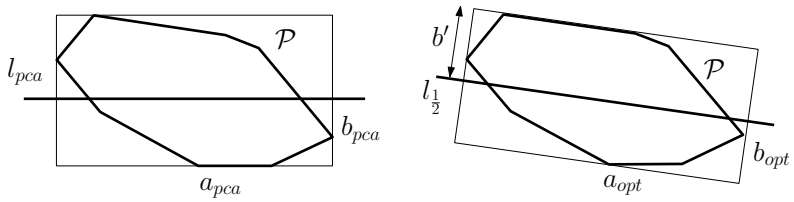


Figure 4: A convex polygon $\mathscr{P}$, its PCA bounding box and the line l_{pca}, which coincides with the first principal component of $\mathscr{P}$, are given in the left part of the figure. The optimal bounding box and the line $l_{\frac{1}{2}}$, going through the middle of its smaller side, parallel with its longer side, are given in the right part of the figure.

general $b_{pca}(P) \leq a_{pca}(P) \leq diam(P) \leq \sqrt{2}a_{opt}(P)$, and in the special case when the optimal bounding box is a square $\lambda_{2,1}(P) \leq 2$. This observation can be generalized, introducing an additional parameter $\eta(P) = a_{opt}(P)/b_{opt}(P)$.

Lemma 8 $\lambda_{2,1}(P) \leq \eta + \frac{1}{\eta}$ *and* $\lambda_{2,2}(P) \leq \eta + \frac{1}{\eta}$ *for any point set P with fixed aspect ratio* $\eta(P) = \eta$.

Proof. We have for both a_{pca} and b_{pca} the upper bound $diam(P) \leq \sqrt{a_{opt}^2 + b_{opt}^2} = a_{opt}\sqrt{1+\frac{1}{\eta^2}}$. Replacing a_{opt} by $\eta \cdot b_{opt}$ in the bound for b_{pca} we obtain $\alpha\beta \leq \eta\left(\sqrt{1+\frac{1}{\eta^2}}\right)^2 = \eta + \frac{1}{\eta}$. □

Unfortunately, this parametrized upper bound tends to infinity for $\eta \to \infty$. Therefore we are going to derive another upper bound that is better for large values of η. In this process we will make essential use of the properties of $BB_{pca(2,1)}(P)$. In order to distinguish clearly between a convex set and its boundary, we will use calligraphic letters for the boundaries, especially $\mathscr{P}$ for the boundary of $CH(P)$ and $\mathscr{BB}_{opt}$ for the boundary of the rectangle $BB_{opt}(P)$. Furthermore, we denote by $d^2(\mathscr{P}, l)$ the integral of the squared distances of the points on $\mathscr{P}$ to a line l, i.e., $d^2(\mathscr{P}, l) = \int_{x\in\mathscr{P}} d^2(x,l)ds$. Let l_{pca} be the line going through the center of gravity and parallel to the longer side of $BB_{pca(2,1)}(P)$ and $l_{\frac{1}{2}}$ be the bisector of $BB_{opt(P)}$ parallel to the longer side. By Lemma 1, part ii) l_{pca} is the best fitting line of P and therefore

$$d^2(\mathscr{P}, l_{pca}) \leq d^2(\mathscr{P}, l_{\frac{1}{2}}). \tag{13}$$

Lemma 9 $d^2(\mathscr{P}, l_{\frac{1}{2}}) \leq \frac{{b_{opt}}^2 a_{opt}}{2} + \frac{{b_{opt}}^3}{6}$.

Proof. If a segment of $\mathscr{P}$ intersects the line $l_{\frac{1}{2}}$, we split this segment into two segments, with the intersection point as a split point. Then, to each segment f of $\mathscr{P}$ flush with the side of the PCA bounding box, we assign a segment identical to f. To each remaining segment s of $\mathscr{P}$, with endpoints (x_1, y_1) and (x_2, y_2), with $|y_1| \leq |y_2|$, we assign two segments: a segment s_1, with endpoints (x_1, y_1) and (x_1, y_2), and a segment s_2, with endpoints (x_1, y_2) and (x_2, y_2). All these segments form the boundary $\mathscr{BB}_S$ of a staircase polygon (see Figure 5 for illustration). Two straightforward consequences are that $d^2(\mathscr{BB}_S, l_{\frac{1}{2}}) \leq d^2(\mathscr{BB}_{opt}, l_{\frac{1}{2}})$, and $d^2(s, l_{\frac{1}{2}}) \leq d^2(s_1, l_{\frac{1}{2}}) + d^2(s_2, l_{\frac{1}{2}})$, for each segment s of $\mathscr{P}$. Therefore, $d^2(\mathscr{P}, l_{\frac{1}{2}})$ is at most $d^2(\mathscr{BB}_S, l_{\frac{1}{2}})$, which is bounded from above by $d^2(\mathscr{BB}_{opt}, l_{\frac{1}{2}}) = 4\int_0^{\frac{b_{opt}}{2}} x^2\,dx + 2\int_0^{a_{opt}} (\frac{b_{opt}}{2})^2\,dx = \frac{{b_{opt}}^2 a_{opt}}{2} + \frac{{b_{opt}}^3}{6}$. □

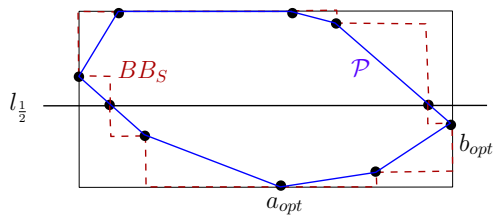


Figure 5: The convex polygon $\mathscr{P}$, its optimal bounding box, and the staircase polygon BB_S (depicted dashed).

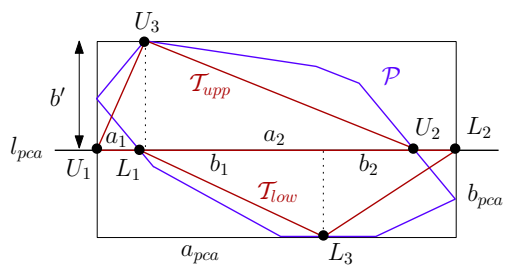


Figure 6: The convex polygon $\mathscr{P}$, its PCA bounding box, and a construction for a lower bound for $d^2(\mathscr{P}, l_{pca})$

 ISBN 978-80-86943-98-5

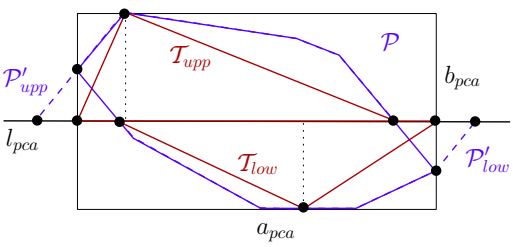


Figure 7: Two polylines $\mathscr{P}'_{upp}$ and $\mathscr{P}'_{low}$ (depicted dashed) formed from $\mathscr{P}$.

Now we look at $\mathscr{P}$ and its PCA bounding box (Figure 6). The line l_{pca} divides $\mathscr{P}$ into an upper and a lower part, $\mathscr{P}_{upp}$ and $\mathscr{P}_{low}$. l_{upp} denotes the orthogonal projection of $\mathscr{P}_{upp}$ onto l_{pca}, with U_1 and U_2 as its extreme points, and l_{low} denotes the orthogonal projection of $\mathscr{P}_{low}$ onto l_{pca}, with L_1 and L_2 as its extreme points. $\mathscr{T}_{upp} = \triangle(U_1U_2U_3)$ is a triangle inscribed in $\mathscr{P}_{upp}$, where point U_3 lies on the intersection of $\mathscr{P}_{upp}$ with the upper side of the PCA bounding box. Analogously, $\mathscr{T}_{low} = \triangle(L_1L_2L_3)$ is a triangle inscribed in $\mathscr{P}_{low}$.

Lemma 10

$$d^2(\mathscr{P}, l_{pca}) \geq d^2(\mathscr{T}_{upp}, l_{pca}) + d^2(\mathscr{T}_{low}, l_{pca}).$$

Proof. Let Q denote a chain of segments of $\mathscr{P}$, which does not touch the longer side of the PCA bounding box, and whose one endpoint lies on the smaller side of the PCA bounding box, and the other endpoint on the line l_{pca}. We reflect Q at the line supporting the side of the PCA bounding box touched by Q. All such reflected chains of segments, together with the rest of $\mathscr{P}$, form two polylines: $\mathscr{P}'_{upp}$ and $\mathscr{P}'_{low}$ (see Figure 7 for illustration). As a consequence, to each of the sides of the triangles $\mathscr{T}_{low}$ and $\mathscr{T}_{upp}$, $\overline{L_1L_3}$, $\overline{L_2L_3}$, $\overline{U_1U_3}$, $\overline{U_2U_3}$, we have a corresponding chain of segments R as shown in the two cases in Figure 8. In both cases $d^2(t, l_{pca}) \leq d^2(R, l_{pca})$. Namely, we can parametrize both curves, R and t, starting at the common endpoint A that is furthest from l_{pca}. By comparing two points with the same parameter (distance from A along the curve) we see that the point on t always has a smaller distance to l_{pca} than the corresponding point on R. In addition t is shorter, and some parts of R have no match on t.

Consequently, $d^2(\mathscr{P}', l_{pca}) \geq d^2(\mathscr{T}_{upp} \bigcup \mathscr{T}_{low}, l_{pca}) = d^2(\mathscr{T}_{upp}, l_{pca}) + d^2(\mathscr{T}_{low}, l_{pca})$, and since, $d^2(\mathscr{P}', l_{pca}) = d^2(\mathscr{P}, l_{pca}) = d^2(\mathscr{P}_{upp} \bigcup \mathscr{P}_{low}, l_{pca})$, the proof is completed. □

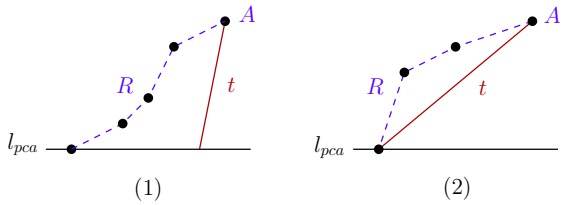


Figure 8: Two types of chains of segments (depicted dashed and denoted by R), and their corresponding triangles' edges (depicted solid and denoted by t).

Since $\mathscr{P}$ is convex, the following relations hold:

$$|l_{upp}| \geq \frac{b'}{b_{pca}} a_{pca}, \text{ and } |l_{low}| \geq \frac{b_{pca} - b'}{b_{pca}} a_{pca}. \quad (14)$$

The value

$$\begin{aligned} d^2(\mathscr{T}_{upp}, l_{pca}) &= \int_0^{\sqrt{a_1^2+b'^2}} \left(\frac{\alpha}{\sqrt{a_1^2+b'^2}} b'\right)^2 d\alpha \\ &+ \int_0^{\sqrt{a_2^2+b'^2}} \left(\frac{\alpha}{\sqrt{a_2^2+b'^2}} b'\right)^2 d\alpha \\ &= \frac{b'^2}{3}\left(\sqrt{a_1^2+b'^2} + \sqrt{a_2^2+b'^2}\right) \end{aligned}$$

is minimal when $a_1 = a_2 = \frac{|l_{upp}|}{2}$. With (14) we get

$$d^2(\mathscr{T}_{upp}, l_{pca}) \geq \frac{b'^3}{3b_{pca}} \sqrt{a_{pca}^2 + 4b_{pca}^2}.$$

Analogously, we have for the lower part:

$$d^2(\mathscr{T}_{low}, l_{pca}) \geq \frac{(b_{pca} - b')^3}{3b_{pca}} \sqrt{a_{pca}^2 + 4b_{pca}^2}.$$

The sum $d^2(\mathscr{T}_{upp}, l_{pca}) + d^2(\mathscr{T}_{low}, l_{pca})$ is minimal when $b' = \frac{b_{pca}}{2}$. This, together with Lemma 10, gives:

$$d^2(\mathscr{P}, l_{pca}) \geq \frac{b_{pca}^2}{12} \sqrt{a_{pca}^2 + 4b_{pca}^2}. \quad (15)$$

Combining (13), (15) and Lemma 9 we have:

$$\frac{1}{2} a_{opt} b_{opt}^2 + \frac{1}{6} b_{opt}^3 \geq \frac{b_{pca}^2}{12} \sqrt{a_{pca}^2 + 4b_{pca}^2} \geq \frac{b_{pca}^2}{12} a_{pca}. \quad (16)$$

Replacing a_{opt} with ηb_{opt} on the left side, b_{pca}^2 with $\beta^2 b_{opt}^2$ and a_{pca} with $\alpha a_{opt} = \alpha\eta b_{opt}$ on the right side of (16), we obtain:

$$\left(\frac{\eta}{2} + \frac{1}{6}\right) b_{opt}^3 \geq \frac{\beta^2 \alpha \eta}{12} b_{opt}^3$$

which implies

$$\beta \leq \sqrt{\frac{6\eta + 2}{\alpha\eta}}.$$

 ISBN 978-80-86943-98-5

This gives the second upper bound on $\lambda_{2,1}(P)$ for point sets with parameter η:

$$\alpha\beta \leq \sqrt{\frac{(6\eta+2)\alpha}{\eta}} \leq \sqrt{\frac{6\eta+2}{\eta}\sqrt{1+\frac{1}{\eta^2}}} \quad (17)$$

Theorem 11 *The PCA bounding box of a point set P in $\mathbb{R}^2$ computed over the boundary of $CH(P)$ has a guaranteed approximation factor $\lambda_{2,1} \leq 2.737$.*

Proof. The theorem follows from the combination of the two parametrised bounds from Lemma 8 and (17) proved above:

$$\lambda_{2,1} \leq \sup_{\eta \geq 1} \left\{ \min \left(\eta + \frac{1}{\eta}, \sqrt{\frac{6\eta+2}{\eta}\sqrt{1+\frac{1}{\eta^2}}} \right) \right\}.$$

It is easy to check that the supremum $s \approx 2.736$ is obtained for $\eta \approx 2.302$. □

5. FUTURE WORK AND OPEN PROBLEMS

It should be possible to prove an upper bound on $\lambda_{2,2}$ along the same line as for $\lambda_{2,1}$, but the analogon of Lemma 9 seems to require some new analytical tools, since, e.g., the reflection tricks do not apply in that setting. However, there is some evidence that an upper bound proof for $\lambda_{2,2}$ would give some ideas to attack the 3-dimensional problem for $\lambda_{3,3}$, and, maybe also a generalization to $\lambda_{d,d}$ in higher dimensions.

REFERENCES

[BCG+96] G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell, and A. Tal. Boxtree: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15:387–396, 1996.

[BHP99] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in 3d. In *10th ACM-SIAM Sympos. Discrete Algorithms*, pages 82–91, 1999.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *ACM SIGMOD Int. Conf. on Manag. of Data*, pages 322–331, 1990.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 1996*, pages 171–180, 1996.

[Jol02] I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 2nd ed., 2002.

[KTR05] H. Kharaghani and B. Tayfeh-Rezaie. A hadamard matrix of order 428. In *J. Combin. Designs 13*, pages 435–440, 2005.

[LKM+00] M. Lahanas, T. Kemmerer, N. Milickovic, D. Baltas K. Karouzakis, and N. Zamboglou. Optimized bounding boxes for three-dimensional treatment planning in brachytherapy. In *Med. Phys. 27*, pages 2333–2342, 2000.

[O'R85] J. O'Rourke. Finding minimal enclosing boxes. In *Int. J. Comp. Info. Sci. 14*, pages 183–199, 1985.

[RL85] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed R-trees. In *ACM SIGMOD*, pages 17–31, 1985.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multidimensional objects. In *13th VLDB Conference*, pages 507–518, 1987.

[Tou83] G. Toussaint. Solving geometric problems with the rotating calipers. In *IEEE MELECON*, May 1983.

[VSR01] D. V. Vranić, D. Saupe, and J. Richter. Tools for 3d-object retrieval: Karhunen-Loeve transform and spherical harmonics. In *IEEE 2001 Workshop Multimedia Signal Processing*, pages 293–298, 2001.

 ISBN 978-80-86943-98-5

Octree-based view-dependent triangle meshes

Marta Fairén
Department of Software
Universitat Politècnica de Catalunya
mfairen@lsi.upc.edu

Ramón Trueba
Department of Software
Universitat Politècnica de Catalunya
rtrueba@lsi.upc.edu

ABSTRACT

In this paper we present a new technique for view-dependent LOD rendering, where the scene is represented through an octree model from which we can obtain a triangle mesh corresponding to a view-dependent LOD. We present the construction of this octree model and the visualization algorithm that generates on-the-fly a closed and valid triangle mesh for each frame of the visualization. This visualization algorithm is a depth-first traversal algorithm which also allows to re-use triangles from one frame to another.

Keywords: View-dependent LOD rendering, octree-based model, valid triangle mesh generation.

1 INTRODUCTION

During the last few decades substantial research efforts have been devoted to devise new techniques to provide interactive navigation through complex 3D models containing thousands of objects and millions of primitives. Most recent approaches fall into three major techniques: level-of-detail (LOD) rendering, image-based rendering, and occlusion culling.

In the case of level-of-detail rendering and more concretely in view-dependent LOD rendering the main problem these techniques have is in those regions where there is a change of level among neighbours, where cracks in the generated triangulation should be avoided.

In our proposal the scene is represented through an octree model which is able to differentiate among objects (allowing further operations like selection, for example, during the navigation) and from which we can obtain a triangle mesh corresponding to a view-dependent LOD. We present the construction of this octree and the algorithm that generates on-the-fly, by doing a depth-first traversal of the octree, a closed and valid triangle mesh for each frame of the visualization.

Some differences between our proposal and other view-dependent LOD techniques are: our data structure does not keep topology information; we do not impose any restriction on the difference of levels among neighbour nodes; and we do keep information about objects.

The main contributions of this paper include:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic

- A new octree-based data structure which encodes the geometry of the scene and which enables the on-line extraction of arbitrary view-dependent LODs.
- An off-line algorithm for building such an octree from different input data.
- An efficient algorithm for rendering the scene encoded by the octree. Triangles extracted from previous frames are efficiently cached and reused on a hierarchical basis.

The rest of the paper is organized as follows. Section 2 briefly reviews the previous work. In section 3 we introduce some definitions and an outline of the algorithm. Sections 4 and 5 explain respectively the definition and generation of the octree model. Section 6 presents the view-dependent octree traversal algorithm. Finally in section 7 we show some results and discussion and we conclude in section 8.

2 PREVIOUS WORK

There are several papers that use hierarchical models to represent very complex scenes and which at the end visualize triangles. Among many of them we can cite [7], which computes a topology preserving isosurface from a volumetric grid using Marching Cubes for geometry processing applications; and [2], which describes an efficient technique for out-of-core construction and accurate view-dependent visualization of very large surface models. It uses a regular conformal hierarchy of tetrahedra to spatially partition the model.

Our approach uses a hierarchical model as well. Similarly to [3], which uses a new spanned sub-meshes simplification technique to build view-dependence trees I/O-efficiently, preserving the correct edge collapsing order and thus assuring the run-time image quality, our octree is view-dependent and we visualize the triangles

that correspond to a front which is being adapted to the point of view at each frame.

Another related paper is [5], which describes a new method for contouring a signed grid whose edges are tagged by Hermite data. They develop an octree-based method for simplifying contours produced by this method. Their method generates quads from Black-White edges. As we will see in the following sections in our case we also consider White-White edges as well as those joining different levels among nodes (T-edges).

Our extraction of triangles from edges is similar to how [6] does to generate triangle strips from the dual mesh.

Schmitt [8] also generates triangles from an octree but in his case the front only moves through complete levels (all nodes in the front should be at the same level).

To the best of our understanding, our approach is the first one that includes a complete view-dependent triangle mesh generation from an octree by doing it on-the-fly and allowing the reuse of geometry already computed.

3 DEFINITIONS AND OUTLINE OF THE VISUALIZATION ALGORITHM

The first step of our proposal consists on a pre-process of input data dedicated to generate the structure that will be used by the view-dependent algorithm. Our input data can be a BRep model or a voxel model. From the input data, the pre-process generates an octree where each grey node contains the coordinates of a point representing the surface inside the node, a normal vector and a colour. Each octree node has binary information of their vertices (*in* or *out*). *Out* vertices will be labeled as white whereas *in* vertices will be labeled as black. The octree specification and generation process is further explained in next section. We will start first by some definitions:

Definition 1 *Relevant edge*
An edge of a node is relevant when it has a white vertex and a black vertex or both vertices are white but the original surface intersects the edge.

Definition 2 *Membrane node*
A membrane node is an octree node containing at least a relevant edge.

Definition 3 *Surface edge*
A relevant edge e is a surface edge of an octree node n ***iff*** *e is one of the 12 edges of n.*

Definition 4 *Edge belongs to node*
A relevant edge e belongs to an octree node n ***iff*** *e is part of any surface edge of n.*

As an example, in figure 1(b) we will say that the red edge is a surface edge of nodes B and D and also belongs to nodes A and C.

Definition 5 *Edge belongs to subtree*
Being S a subtree of the octree and e a relevant edge, e belongs to S ***iff***

$$\forall k : e \text{ belongs to } n_k : n_k \text{ belongs to } S$$

(e belongs to S when the two or four nodes having e as an edge are in the subtree S).

Note: If e belongs to a subtree S, e also belongs to all subtrees containing S.

Definition 6 *Edge internal to node*
A relevant edge e is internal to an octree node n ***iff*** *e belongs to one or more descendants of n but it does not belong to n.*

Definition 7 *T-edge*
A T-edge is a relevant edge which belongs to two membrane nodes and also lays over a face of a third node (see figure 1(a)).

Definition 8 *X-edge*
An X-edge is a relevant edge which belongs to four membrane nodes (see figure 1(b)).

Once we have the octree pre-computed, the algorithm presented in section 6 traverses this structure in a view-dependent manner, by using an *on-the-fly* generation of triangles. The algorithm traverses the octree and for each node being processed it computes the required level in order to decide if the octree level of this node is enough, depending on the point of view, to represent the node in this frame as a leaf node.

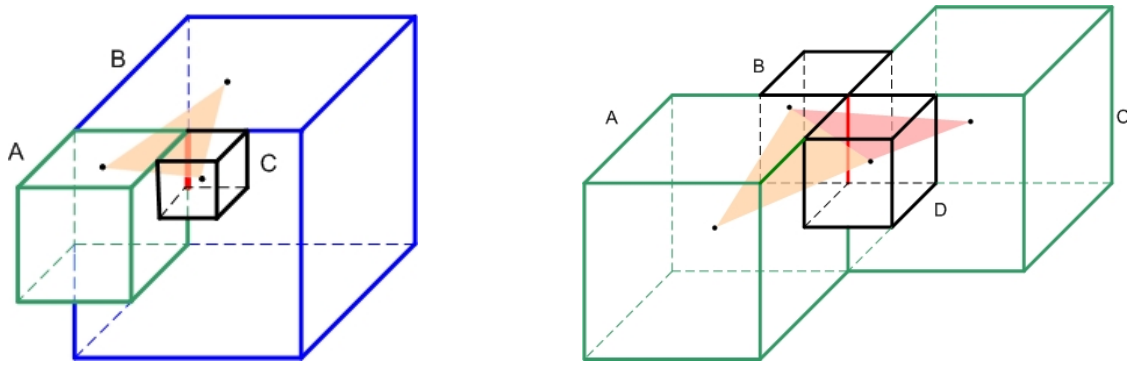
Definition 9 *Front*
We call front to the set of membrane nodes being at their required level.

The *front* is changing at each frame depending on the point of view. The nodes participating on this front are a subset of the membrane nodes. The view-dependent depth-first traversal of the octree stops when a membrane node is in its required level and this node becomes part of the front for this frame.

A white face of a cube representing a membrane node is a face shared with a white node. The union of all white faces of the nodes of the front will be called the white surface of the front.

Taking all *T-edges* and *X-edges* of those nodes participating on the front and generating the corresponding triangles (one for each T-edge and two for each X-edge), it can be shown that we obtain a valid triangulation which represents the scene at the suitable level of detail, where a valid triangulation is a triangulation that is homeomorphic to the white surface of the front.

 ISBN 978-80-86943-98-5



(a) T-edge

(b) X-edge

Figure 1: Example of a T-edge (a) and an X-edge (b). In both cases the edge is the one drawn in red. A T-edge (a) generates a triangle of the final mesh whereas an X-edge (b) generates two triangles. The nearest point to the relevant edge is chosen to divide the quad in the two triangles.

4 OCTREE DEFINITION

The octree model [1] we generate to represent the original model stops its subdivision process when a node does not contain model surface (the node is completely inside or outside the solid); when the node, at this level, already has the necessary information to represent the model surface; or when the subdivision level reaches the maximum level predefined.

With this kind of construction we have a non-restricted octree, where each leaf node can have as neighbours other leaf nodes without any restriction on their size.

For our algorithm, we only need to store information in the grey nodes of the octree, not in the black or white ones. This information includes:

- **A representative point** of the surface inside the node, being represented by its coordinates, a normal vector and a colour.
- **The vertex signs** of the node, indicating, for each one of the 8 vertices of the cube representing the node, whether they are *in* or *out* of the solid.
- **Relevant edges** of the node. We store boolean information indicating which of the 12 edges of the cube are relevant.
- **Object identifier**. We store information about the object whose point is used as a representative point. It facilitates to work with multi-objects.

5 OCTREE GENERATION

Our input data can be a correct BRep model or a voxel model. From this input data, we generate the non-restricted octree containing all the information.

The pre-process consists of two steps. The first step is used to fill the information in the leaf nodes and the second to fill the octree inner nodes. The first step is different depending on the input data.

5.1 Construction from a BRep model

The construction of the octree from a BRep model is done recursively starting from a cube (root node) which includes all the scene, and following a classical subdivision process. This subdivision process stops when the node has the minimum edge length (minimal resolution node) or when the node information fulfills a certain condition (terminal octree node).

The octree nodes are terminal (no more subdivision to do) in two cases:

- when the node doesn't have surface inside it, in this case the node becomes white or black depending on the situation of the node (*in* or *out*) with respect to the solid;
- when the node has solid surface inside it, it becomes terminal if the surface it contains only belongs to an object and fulfills one of the following conditions:
  - it has only one vertex of the geometry;
  - it has only one edge of the geometry;
  - it has only one face of the geometry.

In case the subdivision stops because of a minimal resolution we can have geometry of different objects inside it, so we choose the most relevant object in the node (the one with the largest area inside the node).

The information to be kept in any membrane node is computed as follows:

Representative point coordinates. If the node contains a feature vertex of the solid surface we store

 ISBN 978-80-86943-98-5

the vertex; otherwise, if the node contains a feature edge, we compute that point on the edge nearest to the center of the node; in any other case we compute the nearest point of the surface to the center of the node.

Representative point normal vector. The normal vector stored is the normal of the solid surface at the point we kept as a representative point.

Representative point colour. We take the colour that the representative point has in the original solid.

Signs of vertices. For each node vertex we compute whether it is inside or outside of the solid. The process to compute the signs of vertices is done after the construction of the octree. For each vertex of the cube representing the octree node we compute whether it is white (outside the solid) or black (inside the solid).

Relevant edges. An edge of the node is considered relevant if it has Black-White vertices or it has White-White vertices and the solid surface intersects the edge. This is computed also at the end of the octree construction. We identify those edges having White-White vertices that have to be relevant because the solid surface intersects them. It is also possible to have Black-Black edges intersected by the solid surface, but these edges are not considered relevant because this geometry intersecting the edge is considered internal to the solid and is not visualized.

Object identifier. Identifier of the object whose point is used as a representative point.

5.2 Construction from a voxel model

The construction of the octree from a voxel model starts by identifying each voxel as black, white or membrane node at the minimal resolution of the octree.

The information to be kept in any membrane node at this minimal resolution is filled as follows:

Representative point coordinates. The representative point chosen could be computed through an interpolation from the voxels in the 26-neighbourhood. However, in the present implementation we have chosen to use the middle point of the voxel, because its size is sufficiently small.

Representative point normal vector. We take the gradient of the voxel calculated from the 26-neighbour intensity.

Representative point colour. The colour will be the one kept in the voxel model or computed by using a transfer function from the value of the voxel model [4].

Signs of vertices. We classify a vertex as interior when this vertex does not have any white node around and we classify it as outer when the vertex touches at least one white node.

Relevant edges. An edge of the node is considered relevant if it has Black-White vertices.

Object identifier. Identifier of the object whose point is used as a representative point.

Once we have identified the minimal resolution nodes, we compact those black/white nodes which can be grouped into other black/white nodes to get lower level terminal octree nodes.

5.3 Filling inner nodes

Once all the leaf nodes are processed we have to fill in the octree inner nodes. We apply a bottom-up simplification algorithm and fill each inner membrane node information as follows:

- The sign of each node's vertex corresponds to the sign of the child sharing this vertex with the node.
- The relevant node's edges are those constituted by two White-Black edges or by two White-White edges with at least one of them being relevant.
- The vertex information, normal, colour and object identifier are taken from one of the children elected as representative following the next criterion: A vote is made among the children to discover which object appears in more nodes (the object with a greater surface inside the node) and among the children voting for that object, the one that has the nearest point to the node's center is chosen as the representative.

6 VIEW-DEPENDENT OCTREE TRAVERSAL

The algorithm we use to traverse the octree and generate the view-dependent triangle mesh can be summarized as follows:

 ISBN 978-80-86943-98-5

```
1   advance = true;
2   node = first_son(root);
3   while node != root do
4       if not in_frustum(node) then
5           process_edges_node();
6       else
7           if reusable(node) then
8               Reuse_triangles();
9           else
10              if level_required > current_level and
11                not node.is_terminal() then
12                  node = node.first_son(); // go down
13                  advance = false;
14              else
15                  process_edges_node();
16              endif
17          endif
18      endif
19      if advance then
20          while node != root and
21            node.is_last_son() do
22              node = parent(node); // going up
23              process_edges_non_terminal_node();
24          endwhile
25          if node == root then
26              process_edges_root();
27          else
28              node = next_brother(node);
29          endif
30      endif
31  endwhile
```

The geometry drawn by the algorithm at each frame is computed at run time. The algorithm traverses the octree model and depending on the point of view decides which nodes and at which level will generate the triangles to be drawn (see figure 2).

In order to take profit from the coherence existing between one frame and the next one, the algorithm keeps memory of a list of triangles that stores the triangles drawn in one frame and sorted in the order in which those triangles were generated. This order in the list allows to know each sub-list of triangles belonging to each intermediate node of the octree. When the algorithm decides that a node keeps the same level-of-detail from one frame to the next one, it does not re-compute the triangles for the sub-tree of this node but re-uses those triangles in the list already computed in some previous frame (lines 7 and 8 in the algorithm).

We decide a node is reusable (line 7 in the algorithm) if it fulfills the following conditions:

- it has already generated its corresponding triangles in a previous frame (so these triangles are kept in the list of triangles);
- its level-of-detail has not changed from the previous frame;
- the number of frames passed since the last time the triangles of the node were generated does not exceed a certain predefined limit.

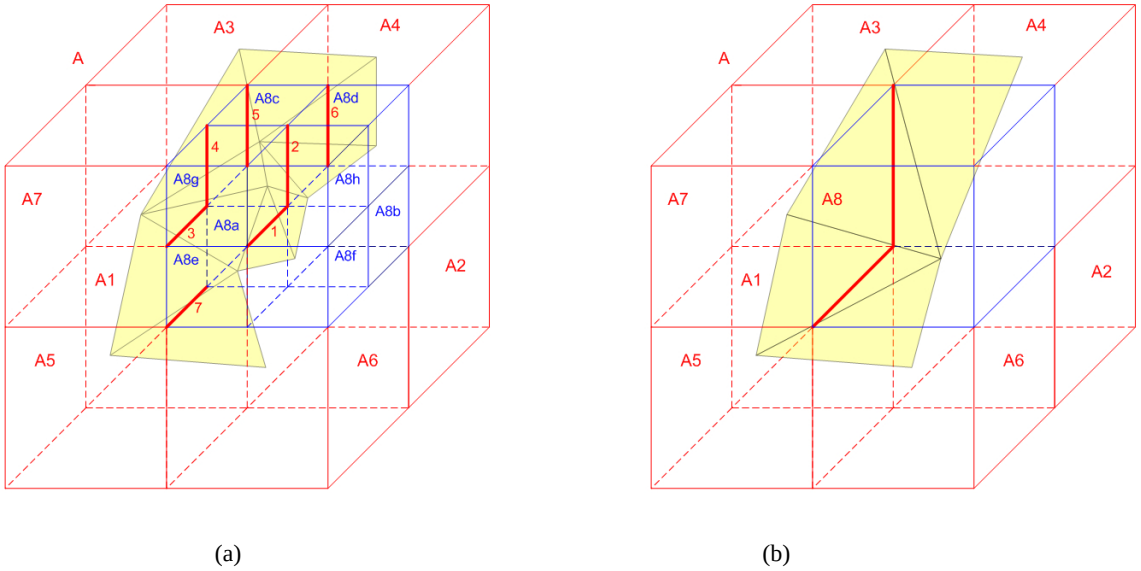
The required level for a node being processed on a frame is computed taking into account the number of pixels that the representation of this node is going to cover on its projection from the user point of view. The level required for the front in a certain subtree depends on the apparent size of their nodes in the screen, which should be smaller than a certain given tolerance. A node being near to the user would require a higher level while a node being far from the user would require a lower level (see figure 3).

The depth traversal of the octree has a treatment done to each node while descending and another done while ascending. While descending, a node can be classified and treated as:

- A node that *can be re-used* from the last frame. The list of triangles corresponding to this node are directly re-used from the last frame (line 8 in the algorithm).
- A node considered *at the required level* for the point-of-view. This node is considered a leaf node (in this frame) and the only treatment required for this node is to recognize those edges of the node that are relevant edges and that should be processed in the treatment to do while ascending (line 15 in the algorithm). These relevant edges will be passed to the parent node to be considered by it (see below the explanation of the process done in the intermediate nodes while ascending).
- A node *not being in the frustum*. Although no treatment would be needed for this node, their edges should be considered in order to generate triangles joining this node with others inside the frustum. It is treated as a node being at the required level (line 5 in the algorithm).
- A node whose required level classifies it as *an intermediate node*. These nodes require no treatment while descending, the algorithm just goes down in the tree (lines 12 and 13 in the algorithm).

While ascending in the depth traversal of the octree, the treatment is mainly centered in the intermediate nodes. Each intermediate node has to process those relevant edges that are internal to the subtree represented by the node (it is the root of the subtree), and which have not been processed before, i.e. it processes those relevant edges that are internal to this node and not to any one of its sons (lines 22 and 23 in the algorithm). The relevant edges that are not internal to the subtree represented by the node will be passed to the parent node. In figure 2(a) the relevant edges 1 and 2 will be

 ISBN 978-80-86943-98-5



(a) (b)

Figure 2: Triangle mesh generated for an example having 7 relevant edges (a) (relevant edges are drawn in thick red line). In case the node A8 is considered at the required level of subdivision (b) only 2 relevant edges are considered and the triangle mesh is simplified.

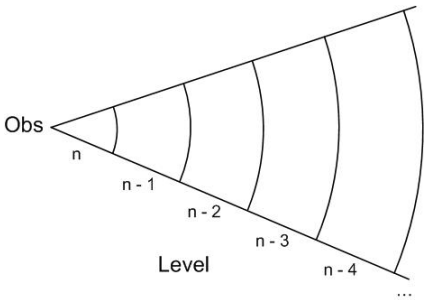


Figure 3: The level for a node is computed depending on the distance to the user.

processed by the node A8, while the relevant edges 3, 4, 5, 6 and 7 will be processed by the root node A (because even though they belong to the A8 subtree, they are not internal to it). The process for each relevant edge consists only on the generation of the triangle (T-edge) or triangles (X-edge) joining the points representing the nodes that are involved in the edge (see figure 1).

7 RESULTS AND DISCUSSION

We have implemented a prototype version of the proposed technique and we have measured its performance with several test models.

The first model is a voxel model of a jaw (see figure 4) containing a total of 1.139.816 voxels. From these, 49.998 are grey voxels (which become membrane nodes at the minimal resolution of the octree). The complete mesh generated from these terminal membrane nodes, without simplification, contains a total of 84.190 triangles.

Figure 5 shows the total number of triangles sent to OpenGL by our algorithm during a short and simple navigation. This navigation consists on moving the camera closer to the jaw and going back. We can observe that when the camera is close to the model the total number of triangles drawn is near to the complete mesh because the level-of-detail should be high enough, while when the camera is going back the number of triangles drawn decreases again.

In figure 6 we can see the number of triangles re-used on each frame during the same short navigation. We can observe that the number of triangles re-used is near to the number of triangles drawn except in those frames where the number of triangles to draw changes (compare this with figure 5 which shows the number of triangles drawn at each frame and also with figure 7 which shows the percentage of triangles re-used). This re-using of triangles already generated can be optimized by using vertex buffer objects of the GPU.

A snapshot of a second model can be seen in figure 7. This model has a total of 2.004.504 voxels with 64.577 of them being grey voxels. The complete mesh generated from these terminal membrane nodes, without simplification, contains a total of 108.534 triangles.

8 CONCLUSION AND FUTURE WORK

We have presented a new technique for a view-dependent LOD rendering, which builds an octree-based data structure enabling the extraction of arbitrary view-dependent LODs and renders the scene by using

 ISBN 978-80-86943-98-5

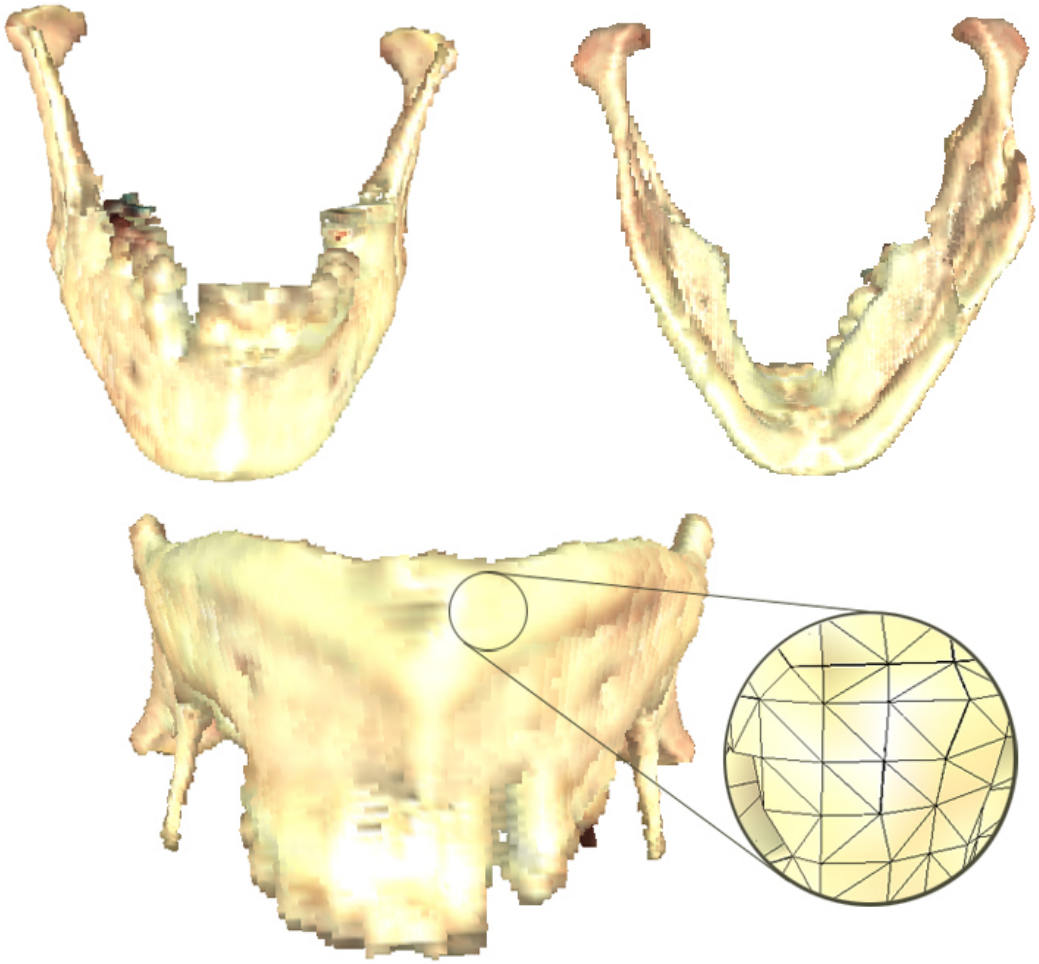


Figure 4: Several views of the model of a jaw during the navigation. The step on the left side of the magnified view is also present in the original model.

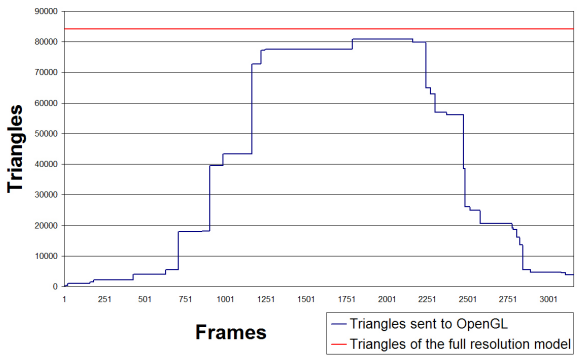


Figure 5: Triangles sent to OpenGL compared with the triangles of the full resolution model.

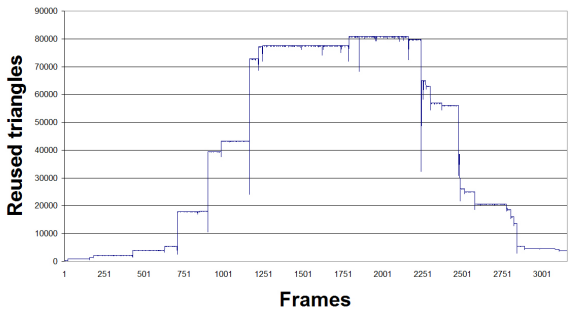


Figure 6: Total number of reused triangles per frame.

a depth-first traversal algorithm which allows to re-use triangles from one frame to another.

As a future work we can improve the efficiency of our current prototype by including the use of vertex buffer objects for the triangles to be re-used among frames and also by generating strips of triangles instead of triangles.

We will also include in the algorithm the possibility of having out-of-core data, so our octree will be stored completed in external memory while having the front nodes kept in internal memory. This will require some

 ISBN 978-80-86943-98-5

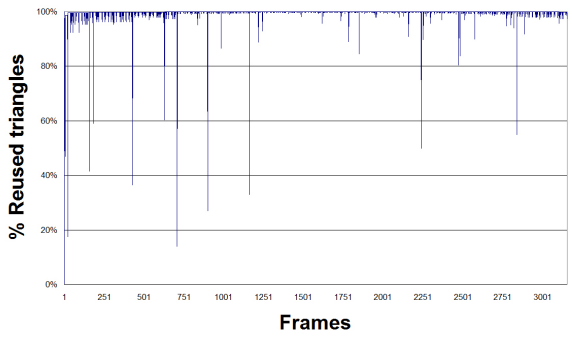


Figure 7: Percentage of reused triangles per frame.

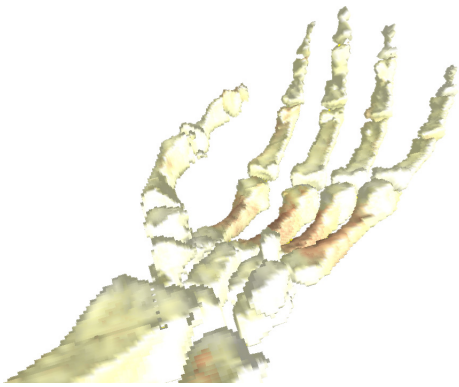


Figure 8: Snapshot of the model of a hand.

pre-fetching techniques to avoid latency on changing the nodes being kept in internal memory.

Finally, another future work is to allow the selection of objects during the navigation, which will permit the user to see an object in more detail than others, for example. To do this, it will be necessary to keep a list of object identifiers at each leaf node of the octree in order to represent correctly those nodes at the minimal resolution.

ACKNOWLEDGEMENTS

We want to thank very specially the collaboration of our colleagues Carlos Andújar, Pere Brunet, Isabel Navazo and Àlvar Vinacua for their valuous help on this work.

This work has been partially supported by the TIN2004-08065-C02-01 project of the Spanish government.

REFERENCES

[1] Pere Brunet and Isabel Navazo. Solid representation and operation using extended octrees. *ACM Transactions on Graphics*, 9(2):170–197, 1990.

[2] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics*, 23(3):796–803, 2004.

[3] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19(3), 2000.

[4] Klaus Engel and Thomas Ertl. Interactive high-quality volume rendering with flexible consumer graphics hardware. In *State of The Art Report. Eurographics'02*, 2002.

[5] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346. ACM Press, 2002.

[6] Massimiliano B. Porcu and Riccardo Scateni. An iterative stripification algorithm based on dual graph operations. In *EuroGraphics 2003 (short presentations)*, pages 69–75, September 2003.

[7] Gokul Varadhan, Shankar Krishnan, T. V. N. Sriram, and Dinesh Manocha. Topology preserving surface extraction using adaptive subdivision. In *SGP'04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 241–250. ACM Press, 2004.

[8] Yucel Yemez and Francis Schmitt. Multilevel representation and transmission of real objects with progressive octree particles. *IEEE Transactions on Visualization and Computer Graphics*, 09(4):551–569, 2003.

Seamless Patches for GPU-Based Terrain Rendering

Yotam Livny
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
livnyy@cs.bgu.ac.il

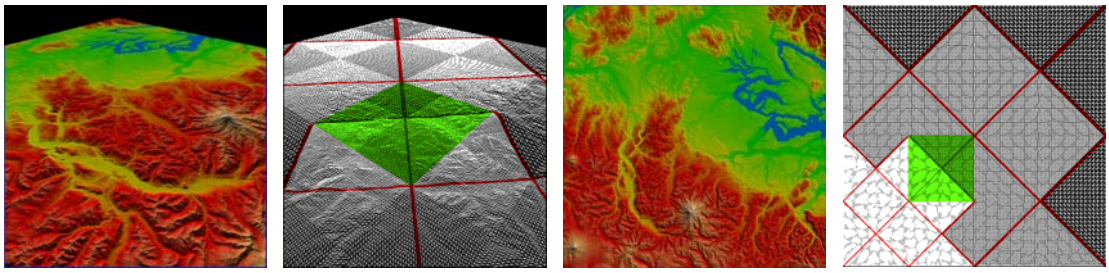
Zvi Kogan
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
koganz@cs.bgu.ac.il

Jihad El-Sana
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
el-sana@cs.bgu.ac.il

ABSTRACT

In this paper we present a novel approach for interactive rendering of large terrain datasets which is based on subdividing the terrain into rectangular patches at different resolutions. Each patch is represented by four triangular tiles which can be at different resolutions; and four strips which are used to stitch the four tiles in a seamless manner. As a result, our scheme maintains resolution changes within patches and not across patches. At runtime, the terrain patches are used to construct a level of detail based on view-parameters. The selected level of detail only includes the layout of the patches and the resolutions at boundary edges. Since adjacent patches agree on the resolution of common edges, the resulted mesh does not include any cracks or degenerate triangles. The GPU generates the meshes of the patches by using scaled instances of cached tiles and assigning elevation for each vertex from the cached textures. Our algorithm manages to achieve quality images at high frame rates while providing seamless transition between different levels of detail.

Keywords: Terrain visualization, view-dependent rendering, and level of detail



(a) (b) (c) (d)

Figure 1: Terrain rendering using seamless patches. (a) A selected view; (b) The wire-frame of (*a*), where the green region marks one patch; (c) Top view of *(a)*; (d) The wire-frame of *(c)* with the same marked patch.

1. INTRODUCTION

Interactive visualization of landscapes and outdoor graphics environments is important for graphics applications such as computer games, flight simulators, and virtual exploration of remote planets. Terrains and height field geometry are vital components of these virtual environments.

The rapid development in acquisition of topographic maps and cartography has led to the generation of large terrain datasets that contain billions of samples. Such terrains exceed the rendering capability of available graphics hardware, thus reducing the geometric complexity of these datasets is mandatory for interactivity. Adjusting the terrain triangulation in a view-dependent manner is a common approach for interactive terrain rendering. Furthermore, adaptive level-of-detail rendering not only simplifies the geometry, but also manages to reduce aliasing artifacts that may result from rendering uniform dense triangulation.

The challenges of interactive terrain rendering have attracted the interest of researchers for several decades and extensive research has been done (see Section 2). Classic level-of-detail rendering schemes generate, usually off-line, multiresolution hierarchies which are used at runtime to guide the selection of appropriate levels of detail based on view-parameters. Some of these approaches utilize temporal coherence among consecutive frames by adaptively simplifying or refining the geometry of a frame to produce the next frame's geometry. Other approaches generate the geometry for each frame independent of its previous frames. These approaches have managed to accelerate the rendering of large terrains, but they were not able to maintain the improvement rate as the GPUs grow faster. In addition, generation of the frame's geometry is performed by executing refine and simplify operations on the CPU, which often fails to complete these computations within the duration of one frame. This geometry, which is transferred to the graphics hardware at each frame, often exceeds the bandwidth

 ISBN 978-80-86943-98-5

of the communication channel and results in unacceptably low frame rates.

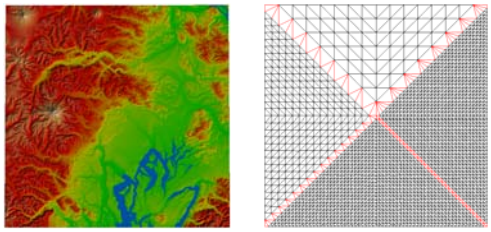


Figure 2: An image of one patch (left) and its wire-frame showing the stitching strips (right).

To reduce computation load on the busy CPU, several approaches partition the terrain into patches at different resolutions. At runtime these patches are stitched together to generate the appropriate levels of detail, which are then transmitted to the graphics hardware. Stitching these patches in a seamless manner is the main challenge for these approaches. Introducing degenerate triangles and dependencies among patches are used to handle these problems. However, these solutions may introduce visual artifacts or require additional random-access memory references.

To reduce data transmission between CPU and GPU, several algorithms use cached templates and quadric terrain elevation maps to generate geometry within the GPU. These algorithms often rely on triangular templates, which do not fit the rectangular texture interfaces and, hence, impose additional complexity in maintaining and storing textures.

In this paper we present a novel approach for interactive rendering of large terrain datasets, which is designed to prevent the above limitations of previous algorithms. Our approach subdivides the terrain into rectangular patches at different resolutions as shown in Figure 1. Each patch is represented by four triangular tiles that can be at different predetermined discrete resolutions and are stitched together by four strips as shown in Figure 2. Since the number of different resolutions is very small, the number of required patterns of stitching strips is also very small.

At runtime, these patches are used to construct the appropriate level of detail based on view-parameters.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

The selected levels of detail do not include any geometry; instead they only include the layout of the patches and the resolutions along their boundaries. The resolutions along the boundaries are used to guide the selection of the adequate tiles and strips to cover each patch without the need to query adjacent patches. Since adjacent patches agree on the resolution of the shared edges, the generated mesh does not include any cracks or degenerate triangles. Scaled templates of the cached tiles are used to generate the geometry, within the GPU, based on the boundary resolution. The vertex and fragment processors fetch and assign elevation and color for each vertex using the cached textures. To handle large terrain datasets, we provide external texture memory support that caches the necessary displacement and color maps in the GPU's memory.

Our approach provides a number of advantages over previous terrain rendering schemes. The level of detail in each patch is determined without querying adjacent patches. Such a scheme saves unnecessary random-access memory references. The rendered mesh does not include any degenerate or sliver triangles, since our approach assures the same triangulation on the two sides of each boundary edge. In addition, it uses an implicit hierarchical representation that maintains the structure of the different patches in runtime. Furthermore, it reduces communication overhead as a result of transmitting only the layout of patches to the GPU at each frame, and using predetermined planar triangular tiles, which are cached in texture memory, to generate the selected level-of-detail representation. Therefore, only elevation values are transmitted to the graphics hardware in each frame.

In the rest of this paper we briefly overview related work in terrain rendering. Then we discuss our novel approach, followed by implementation details and experimental results. Finally, we draw some conclusions and suggest directions for future work.

2. RELATED WORK

In this section we briefly overview related work in level-of-detail terrain rendering. We focus on approaches that utilize the special properties of height-field datasets.

General level-of-detail rendering algorithms represent terrains as triangulated meshes. They usually utilize temporal coherence and manage to achieve the best approximation of the terrain for given view-parameters and triangle budget. However, these algorithms require the maintenance of mesh adjacency and validation of refinement dependences at each frame.

Level-of-detail algorithms for height-field datasets are based on regular grid representation. They utilize

 ISBN 978-80-86943-98-5

the longest edge bisection scheme to simplify memory layout by using a restricted quadtree triangulation [Bao04a, Paj98a], triangle bintrees [Duc97a, Lin96a], or hierarchies of right triangles [Eva01a, Lin02a]. However, updating the mesh at each frame prevents the use of efficient rendering schemes, such as geometry caching.

To utilize efficient rendering schemes, several approaches partition the terrain into square patches at different resolutions. At runtime the appropriate patches are selected, stitched together, and rendered [Hit93a, Paj98a, Pom00a]. Cignoni *et al*. [Cig04a] and Yoon *et al.* [Yoo05a] have developed similar approaches for general 3D models. The main challenge for these approaches is to stitch the boundaries of the appropriate patches seamlessly.

To overcome this communication bottleneck several algorithms have utilized cached geometry. Various approaches cache triangulated regions in texture memory [Cig03a, Cig03b, Lar03a, Lev02a], while others exploit the geometric locality to maximize the efficiency of the cache [Hop99a]. Terrains usually compensate small geometric details by textures and as a result, they are often accompanied by huge texture maps. Tanner *et al*. [Tan98a] have introduced the texture clipmaps hierarchy, and Döllner *et al*. [Dol00a] have developed a more general hierarchy to handle large texture maps. Caching techniques enable fast transfer of geometry and texture to graphics hardware. However, cache memory is limited, thus large datasets may still involve an overhead in communication between CPU and cache memory.

Cook [Coo84a] introduced the displacement maps that represent elevation maps as vertex textures. Other frameworks for displacement maps on programmable graphics hardware have been suggested by [Dog00a, Gum99a, Los04a, Mou02a]. Although these approaches are not implemented at GPU, they are based on designs which prefer many simple computations over a few complicated ones.

The advances in graphics hardware and its programmability have driven the development of a new generation of level-of-detail rendering algorithms. Losasso *et al*. [Los03a] and Bolz and Schröder [Bol05a] used the fragment processor to perform mesh subdivision. Southern and Gain [Sou03a] and Larsen and Christensen [Lar03a] used the vertex processor to interpolate different resolution meshes in a view-dependent manner. Wagner [Wag04a] and Hwa *et al*. [Hwa04a] used GPU-based geomorphs to render terrain patches of different resolutions. Dachsbacher and Stamminger [Dac04a] used GPU programmability to generate and render procedural details for terrains at runtime. Schneider and Westermann [Sch06a] suggested progressive transmission to reduce the data transfer between CPU and GPU.

Geometry clipmaps algorithm [Asi05a] stores the surface triangulation layout in a view-dependent manner. In each frame, the visible part of the triangulation is sent to the GPU and modified according to the uploaded elevation and color maps. However, this algorithm does not perform local adaptivity, and the transition between levels of detail is not smooth and may result in cracks. The cracks problem is resolved by inserting degenerate triangles, but such triangles may generate visual artifacts.

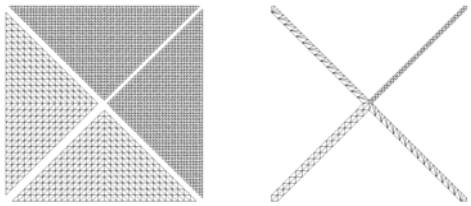
3. OUR APPROACH

In this section we present our novel algorithm for interactive terrain rendering. It partitions the terrain into rectangular patches and utilizes advanced features of graphics hardware, such as programmability, displacement mapping, and geometry caching. Our algorithm involves a light preprocessing stage, in which it generates hierarchies of elevation maps and color textures and stores them in main memory. In our patch scheme, the coexistence of different discrete geometry resolutions within the same patch enables seamless stitching (without cracks or degenerate triangles) of adjacent patches. In each frame our algorithm uses an implicit patch hierarchy to select a set of appropriate patches (for rendering) and determine the resolution on their boundaries based on view parameters. The resolution of each patch is determined based on its four boundary edges and without the need to query its adjacent patches.

Patch Scheme

Previous terrain rendering algorithms use either triangular or rectangular patches for view-dependent level-of-detail rendering. On one hand, algorithms that use rectangular patches assign constant resolution over the entire patch, and hence prevent local adaptivity and impose severe difficulty in stitching adjacent patches. On the other hand, algorithms that use triangular patches enable easier stitching schemes and provide better local adaptivity, but they suffer incompatibility with texture rectangular interface and complicate texture management. Our patch scheme combines the advantages of the two approaches; it subdivides the terrain into rectangular patches which consist of triangular tiles that allow different resolutions to coexist within one patch. Such a scheme provides limited local adaptivity and enables the stitching of adjacent patches in a seamless manner.

 ISBN 978-80-86943-98-5



(a) (b)

Figure 3: The components of one patch. (a) The image of four tiles. (b) The image of four strips.

In our scheme, a patch is arranged as four tessellated triangular regions which are determined by the two diagonals of the rectangular patch (see Figure 3). We shall refer to these tessellated triangular regions as *triangular tiles* (or simply *tiles*). The four tiles can have different resolutions which are selected from a predefined set of uniform resolutions. One could treat these tiles as discrete levels of detail of the same tile. Within a patch, the triangular tiles are stitched together by using predefined strips (refer to Figure 4). Since the number of different resolutions for the tiles is usually small – *2* to *4* – the number of different stitching strips is also very small. Six strip types are required to stitch tiles at three different resolutions.

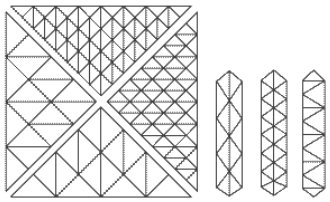


Figure 4: Triangular tiles at two different resolutions and the required stitching strips.

We have chosen to adopt tile resolutions at consecutive powers of two to comply with the mipmap resolutions and meet the requirement of *Claim 1* (see Level of Detail section below).

Patch Hierarchy

The patch hierarchy is constructed top-down by subdividing each patch into $R \times R$ children patches, where R is the branching factor of the hierarchy. The branching factor is determined by the number of different resolutions for tiles and equal to the ratio between the smallest and largest resolutions. For example, *2* and *3* resolutions require branching factors of *2* and *4*, respectively. This relation ensures seamless stitching among adjacent patches and absence of cracks.

The patch hierarchy does not store any geometry; instead it stores the position and dimension of each patch with respect to the terrain. Therefore, it easily fits in local memory, even for very large terrains. In practice, there is no need to implement the hierarchy explicitly, and therefore in our current implementation we use implicit hierarchy.

Runtime Rendering

At runtime, the patch hierarchy is used to guide the selection of the various levels of detail based on view-parameters. In each frame, the patch hierarchy is traversed in a top-down manner to select a set of active patches that form the appropriate level of detail. The traversal process starts from the root and for each visited patch τ an error metric is computed. If the error is too large, with respect to the view-parameters, the children of the patch τ are traversed. Otherwise, the resolutions of boundary edges are computed and the patch is added to the stream of active patches.

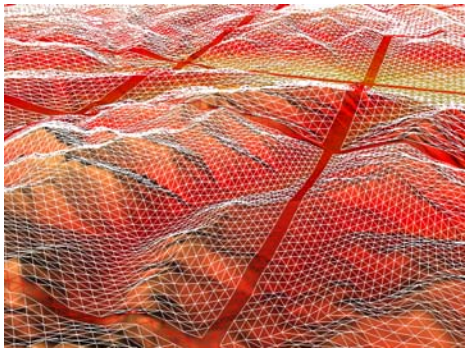


Figure 5: A terrain view with a wire-frame on top. The meshes of triangular tiles appear in white color and the strips appear in red.

Single-unit-size meshes that represent each resolution of the tiles and stitching strips (3 tiles and 6 strips for three different resolutions) are cached in texture memory. In each frame, the active patches are streamed to the graphics hardware for rendering. The light representation of active patches by their 2D enclosing rectangle contributes to the dramatic reduction on the CPU-GPU communication load. The resolutions at boundary edges (of patches) are discretized to match the resolution of the predefined triangular tiles. The resolution at the boundary edges is enough to determine the tiles and strips required to cover the patch τ in a straight forward manner (see Figure 5). The cached instances of the selected tiles and strips are transformed to match the enclosing rectangle of the patch, which selects its tiles without querying any of its adjacent patches. Since each two adjacent patches agree on the resolution of the common edge, the stitching of adjacent patches is smooth and does not include cracks or degenerate triangles.

Tiling a patch with triangular tiles produces a planar mesh without elevation or color components. These components are assigned (for each vertex) by the

 ISBN 978-80-86943-98-5

vertex and fragment processors, which use *x* and *y* coordinates of a received vertex to fetch and assign the appropriate elevation/color from cached textures.

Level of Detail

The level of detail of a patch is represented by the resolutions of its tiles which are determined by the resolution at boundary edges. The resolution of an edge is computed based on its length *l* and the distance *d* from the viewpoint by using Equation 1, where ρ is a precision factor. If ε is larger than 1, the patch is split to its children, otherwise the resolution of the edge is determined by εR_{max} rounded up to the closest resolution, where R_{max} is the highest available resolution.

$$\varepsilon = \rho \frac{l}{d} \tag{1}$$

The scaling factor is used to resize a tile to match the patch's enclosing rectangle and select the appropriate texture level from which the elevation and color values are fetched.

Claim 1: *The generated mesh does not include cracks, which means that any two adjacent tiles agree on the resolution of the common edge.*

Proof: Without loss of generality we prove the claim for two resolutions and quadtree subdivision. We distinguish between two cases:

I. The two adjacent patches have the same dimensions: Since the two patches have the same dimensions, they have the same enclosing rectangle and share a common edge along an entire side. By selecting the same tile on the two sides of the shared edge, the two patches are stitched seamlessly.

II. The two adjacent patches are in different dimensions, which means that the edge belongs to one patch on one side and two patches on the other side (see the edge $\overline{AB}$ in Figure 6): We first show that the tile $\overline{ABJ}$ gets the highest resolution *R2*. The patch $\overline{ABCD}$ has split to four children, which means that one of its edges has required a resolution higher than *R2* (beyond the available resolution); let this edge be $\overline{CD}$. Let *l* be the length of the edge $\overline{AB}$ and the distances of the edges $\overline{AB}$ and $\overline{CD}$ from the viewpoint are d_{far} and d_{near}, respectively. Based on Equation 1, $\rho \cdot l > d_{near}$ holds as a result of assuming that the patch $\overline{ABCD}$ has split into its four children, then:

$$\frac{\rho \cdot l}{d_{near}} > 1 \Rightarrow \frac{\rho \cdot l}{l + d_{near}} > \frac{1}{2} \Rightarrow \frac{\rho \cdot l}{d_{far}} > \frac{1}{2}; d_{far} \le l + d_{near}$$

Therefore, the edge $\overline{AB}$ is assigned the resolution *R2*, and the edges $\overline{AE}$ and $\overline{EB}$ are assigned the resolution *R1*. Our algorithm assigns resolution *R1* to edges with error values ε in the range *[0, 0.5]* and *R2* to those with error values in the range *(0.5, 1.0]*. For that reason, the difference between adjacent patches is at most one level (in the case of two different resolutions). ■

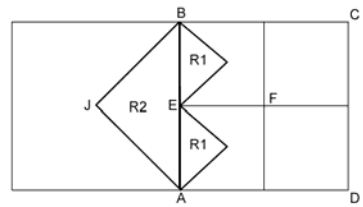


Figure 6: Stitching tiles at two different levels of detail.

Texture Pyramid

Terrain datasets are usually represented by elevation maps and color textures, which store the properties of vertices in the original terrain. We use multiple-level texture pyramids at successive powers of two (similar to mipmaps) to support level-of-detail rendering. These texture pyramids are used at runtime to achieve faithful sampling of the textures for the vertices of each tile. Since these multiple-level pyramids are similar to mipmaps, we could let the hardware construct them. Then at runtime, the vertex processor determines from which level to select the values. However, such an approach does not work when the terrain size exceeds the capacity of the base level of the mipmaps [Los04a].

For large terrains, the multiple-level texture pyramids are constructed once by the CPU before being transferred for caching in the texture memory. We start with the original texture, which represents the most detailed level, and each new level is generated from the previous one by reducing the resolution by half at each dimension. The pixels in the generated level are computed by interpolating the four corresponding pixels of the previous level.

Note that elevation and color values of a vertex are selected from different levels of the hierarchies based on the geometric level of detail of the processed tile.

4. IMPLEMENTATION DETAILS

In our current implementation we do not construct the patch hierarchy explicitly; instead, an implicit representation is used. The root of the hierarchy is the coarsest level of detail that fits in texture memory and matches the interactive rendering capability of the graphics hardware. Therefore, the height of the hierarchy can be easily determined based on the hardware capabilities and the predefined branching factor. Note that the 2D bounding rectangle of the root is the same as that of the original terrain. Also

 ISBN 978-80-86943-98-5

Dataset Size	View Region	ρ Factor	With Frustum Culling					Without Frustum Culling			
			Triangles	Traversed	Rendered	Culled	FPS	Triangles	Traversed	Rendered	FPS
4Kx4K	Edge	ρ_0	172696	109	46	36	283	331428	121	91	156
4Kx4K	Middle	ρ_0	138668	109	33	49	380	403896	141	106	138
16Kx16K	Edge	ρ_0	180731	100	50	27	271	338240	123	93	153
16Kx16K	Middle	ρ_0	148200	126	39	58	354	389197	151	107	135
4Kx4K	Edge	$2\rho_0$	354248	137	66	37	138	763864	189	142	69
4Kx4K	Middle	$2\rho_0$	330480	133	56	44	156	1133796	265	199	52
16Kx16K	Edge	$2\rho_0$	422358	179	82	74	112	915190	213	160	56
16Kx16K	Middle	$2\rho_0$	414966	152	70	63	112	1359642	340	231	43

Table 1: Runtime performance.

recall that the patch hierarchy does not store any mesh geometry or pixel information. The subdivision of a patch into its children is performed by several shift instructions within the CPU. The traversal of the implicit patch hierarchy is performed similar to the explicit one and is often more efficient as a result of avoiding random memory access to fetch the children patches. We found that traversing the patch hierarchy is negligible compared to the rendering time as shown in the CPU column on Table 2.

View-frustum culling is performed by the CPU during the traversal which determines the set of active patches. For each patch τ which requires further subdivision to reach the appropriate level of detail, we test its children patches against the view-frustum only if τ intersects the boundary of the view-frustum. If the patch τ is entirely included within the view-frustum, then all its children patches are also within the view-frustum. If τ intersects the view-frustum's boundary, we test and mark each of its children patches as to whether it is inside, outside, or intersecting the view-frustum. Outside patches are culled and they are not processed further.

The meshes that represent the different resolutions tiles and strips are cached in texture memory. At runtime, these meshes are used to tile the selected patches. Since the number and the size of these meshes are small (3 tiles and 6 strips are required to support three different resolutions within a patch), we store four orientations of each tile and each strip to avoid rotation and mirroring of these meshes at runtime.

To handle large terrain datasets we have implemented an out-of-core support similar to the one proposed by Losasso and Hoppe [Los04a]. This scheme stores the texture pyramids in main memory and caches in texture memory only the portions necessary for rendering. The updates of the cached textures are performed in an active manner by loading "L-Shape" regions into texture memory, as early as they are required.

In earlier algorithms, the CPU needs to send three coordinates for an uncached vertex to place it in the model space. Our algorithm utilizes hardware supported displacement mapping, and thus the CPU sends only the elevation value for each vertex. The other two coordinates are generated by the GPU in a parametric manner using the terrain grid structure. This technique reduces the data transfer at runtime from three coordinates to one coordinate for each vertex. The elevation components are uploaded into the vertex texture using Fragment Buffer Object extensions (FBO).

5. RESULTS

We have tested our implementation on an AMD Athlon 3500 with 1GB memory, and an *nVidia* GeForce 7800 GTX graphics card with 256M texture memory using Puget Sound and Grand Canyon terrain datasets. In this section we report and analyze selected entries of these results.

The performances of our algorithm are summarized in Table 1. For each dataset we view different regions of the terrain to capture the various processing patterns. In each row we report the terrain size, viewed region, precision factor, and performance with and without view-frustum culling. We record two options for the viewed regions: *edge* and *middle,* which refer to flying near an edge and inside the terrain, respectively. The precision factor (see also Equation 1) $2\rho_0$ selects more detailed levels than the levels selected by ρ_0. In the performance columns we report the number of rendered triangles (*Triangle* column), the number of traversed patches (*Traversed* column), the number of rendered patches (*Rendered* column), the number of culled patches (*Culled* column), and the frame rates. The view-frustum culling doubles the performances when flying on the edge of the terrain and triples it in general. Our algorithm manages to achieve quality images at high frame rates, as can be seen in Table 1. The frame rates depend mainly on the number of triangles. The first row shows 156 FPS without view frustum culling for about 330K triangles and 91 rendering patches, and the sixth row reports the same FPS with view frustum culling and 56 patches. Such observation reveals that patch selection is negligible with respect to the total rendering time. Note that

 ISBN 978-80-86943-98-5

patch selection also includes view-frustum culling and transmission of active patches list.

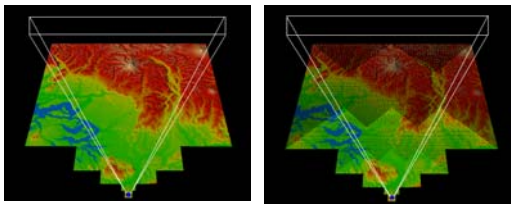


Figure 7: View-frustum culling: A shaded view (left) and its wire-frame representation (right).

We compared the results of our algorithm with the results of three known terrain rendering algorithms. To bring all the results to a common base, we have estimated the expected performance of these algorithms on our machine based on the reported results and the used machine's hardware. To present a reliable approximation, we measure only triangles that are actually processed by the graphics hardware. On comparable hardware we expect that BDAM [Cig03a], Clipmap [Asi05a], and the algorithm suggested in [Hwa04a] will achieve about *46M*, *44M*, and *43M* textured triangles per second, respectively. Our algorithm manages to achieve *53M* textured triangles per second on average. These numbers show that the simplicity of our GPU code with the advantages of displacement map functionality provides encouraging performance.

Configuration			Time		
Triangles	Rendered	Culled	CPU (μs)	GPU (ms)	FPS
88986	18	34	17.54	1.85	583
96246	22	13	22.57	1.98	545
119054	27	18	26.88	2.58	418
148200	39	58	30.67	3.04	354
180731	50	27	34.84	3.98	271
230372	47	23	39.06	5.68	190

Table 2: Hardware performance analysis

The contribution of the CPU and the GPU to the performance of the algorithm is shown in Table 2. The first three columns of each row represent the configuration of a frame, which includes the number of rendered triangles, rendered patches, and culled patches. The fourth and fifth columns report the CPU and the GPU processing time, respectively. The CPU load is tiny and has almost no influence on the frame rates for two main reasons – the selection of the active patches (by the CPU) is very light and the CPU runs parallel to the GPU. These conclusions are also supported by the results shown in Table 2. These results also show that our algorithm will benefit from the current trend in improving GPU rates.

Figure 7 shows the shaded and wire-frame representations of a terrain view after applying view-frustum culling. Figures 8 and 9 were generated from a Puget Sound terrain dataset using our algorithm at different precision factors ρ. In each figure, image (a) shows a shaded view that depicts image quality, image (b) shows the wire-frame representation that illustrates the triangular tiles in white color and the stitching strips in red.

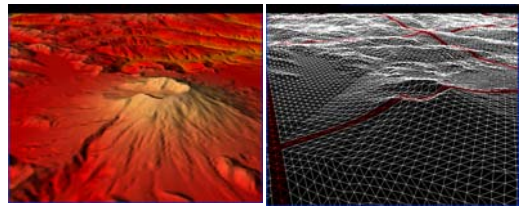


Figure 8: A terrain view at $\rho = \rho_0$. (a) A shaded surface. (b) Tiles in white and strips in red.

6. SUMMARY AND FUTURE WORK

We have presented a novel approach for interactive terrain rendering that reduces the load on the CPU, utilizes texture memory, and leverages advanced features of the GPU. The terrain is subdivided into rectangular patches on the fly. Each patch is represented by four triangular tiles at different resolutions which are stitched together using four strips. At runtime the CPU selects the appropriate patches based on view-parameters and determines the resolution at their boundaries. The different tiles and stitching strips are cached in texture memory and used to tile each patch according to its boundary resolution. Multiresolution levels of color textures and displacement maps are also cached in texture memory and used by the vertex and fragment processors to assign the elevation and color for each vertex.

Our approach balances computation load among the CPU and GPU and dramatically reduces the communication between them. Adjacent patches are stitched in a seamless manner without cracks or degenerate triangles, since they agree on the resolution of the common edge. Furthermore, each patch determines its own resolution without querying its adjacent patches; it simply selects the different tiles that comply with its boundary resolution. The use of tiles provides limited local adaptivity which contributes to the smoothness of the generated mesh.

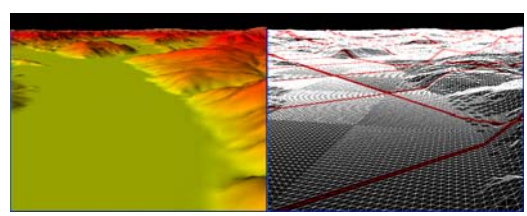


Figure 9: A terrain view at $\rho=2\rho_0$. (a) A shaded surface. (b) Tiles in white and strips in red.

Our algorithm performances are strongly influenced by the speed of vertex pipelines. The algorithm relies

 ISBN 978-80-86943-98-5

on the vertex fetch operation which enables the vertex processor to access texture memory. However, the fetch operations within vertex processors are not yet optimized. We predict that future development in vertex processor hardware will lead to impressive improvement on the performance of our algorithm.

We see the scope of future work in extending the idea of independent patches to general 3D models. Such development will provide view-dependent rendering for large datasets in a seamless manner without imposing dependencies among adjacent patches. Moreover, our suggested approach generates patch geometry within the GPU, and hence can not utilize temporal coherence among consecutive frames. Utilizing temporal coherence within the GPU could contribute to further performance improvements.

7. ACKNOWLEDGMENTS

Our research and particularly this work are supported by the Lynn and William Frankel Center for Computer Sciences, the Israel Ministry of Science and Technology, and the Tuman fund.

8. REFERENCES

[Asi05a] Asirvatham A. and Hoppe H. Terrain rendering using GPU-based geometry clipmaps. GPU Gems 2, pages 27–45, 2005.

[Bao04a] Bao X., Pajarola R., and Shafae M., Smart: An efficient technique for massive terrain visualization from out-of-core. In Proceedings of Vision, Modeling and Visualization ’04, pages 413–420, 2004.

[Bol05a] Bolz J. and Schröder P. Evaluation of subdivision surfaces on programmable graphics hardware. Submitted, 2005.

[Paj98a] Pajarola R., Large scale terrain visualization using the restricted quadtree triangulation. In Proceedings of Visualization ’98, pages 19–26, 1998.

[Cig03a] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. Computer Graphics Forum, 22(3):505–514, 2003.

[Cig03b] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. Planet-sized batched dynamic adaptive meshes. In Proceedings of Visualization ’03, pages 147–155, 2003.

[Cig04a] Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. ACM Trans. Graph., 23(3):796–803, 2004.

[Coo84a] Cook L., Shade trees. Computer Graphics, 18(3):223–231, 1984.

[Dac04a] Dachsbacher C., and Stamminger M. Rendering procedural terrain by geometry image warping. In Eurographics Symposium in Geometry Processing, pages 138–145, 2004.

[Dog00a] Doggett M., and Hirche J. Adaptive view dependent tessellation of displacement maps. In HWWS ’00: Proceedings of the workshop on Graphics hardware, pages 59–66, 2000.

[Dol00a] Döllner J., Baumann K., and Hinrichs K. Texturing techniques for terrain visualization. In Proceedings of Visualization ’00, pages 227–234, 2000.

[Duc97a] Duchainear M., Wolinsky M., Sigeti D., M. Miller, C. Aldrich, and Mineev-Weinstein M., ROAMing terrain: Real-time optimally adapting meshes. In Proceedings of Visualization ’97, pages 81–88, 1997.

[Eva01a] Evans S., Kirkpatrick D., and Townsend G. Right triangulated irregular networks. Algorithmica, 30(2):264–286,2001.

[Lin96a] Lindstrom P., Koller D., Ribarsky W., Hodges L., Faust N., and Turner G. Real-time, continuous level ofdetail rendering of height fields. In SIGGRAPH ’96,pages 109–118, 1996.

[Lin02a] Lindstrom P. and Pascucci V. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. IEEE Transactions on Visualization and Computer Graphics, 8(3):239–254, 2002.

[Gum99a] Gumhold S. and H¨uttner T. Multiresolution rendering with displacement mapping. In HWWS ’99: Proceedings of the workshop on graphics hardware, pages55–66, 1999.

[Hit93a] Hitchner L. and McGreevy M. Methods for user-based reduction of model complexity for virtual planetary exploration. In SPIE 1913, pages 622–636, 1993.

[Hop99a] Hoppe H. Optimization of mesh locality for transparent vertex caching. In SIGGRAPH ’99, pages 269–276, 1999.

[Hwa04a] Hwa L. M., Duchaineau M., and Joy K. I.. Adaptive 4-8 texture hierarchies. In Proceedings of Visualization 2004, pages 219–226, 2004.

[Lar03a] Lario R., Pajarola R., and Tirado F. Hyper-block quadtree based triangulated irregular networks. In Proceedings of IASTED VIIP, pages 733–738, 2003.

[Lev02a] Levenberg J. Fast view-dependent level-of-detail rendering using cached geometry. In Proceedings of Visualization ’02, pages 259–266, 2002.

[Los03a] Losasso F., Hoppe H., Schaefer S., and Warren J. Smooth geometry images. In Eurographics Symposium in Geometry Processing, pages 138–145, 2003.

[Los04a] Losasso F. and Hoppe H. Geometry clipmaps: terrain rendering using nested regular grids. ACM Trans. Graph., 23(3):769–776, 2004.

[Mou02a] Moule K. and McCool M. Efficient bounded adaptive tessellation of displacement maps. In Proceedings of Graphics Interface, pages 171–180, 2002.

[Pom00a] Pomeranz A. Roam using triangle clusters (RUSTiC). Master’s thesis, 2000.

[Sch06a] Schneider J. and Westermann R. Gpu-friendly high-quality terrain rendering. Journal of WSCG, 14(1-3):49–56, 2006.

[Sou03a] Southern R., and Gain J., Creation and control of real-time continuous level of detail on programmable graphics hardware. Computer Graphics Forum, 22(1):35–48, 2003.

[Tan98a] Tanner C., Migdal J. and Jones M. The clipmap: a virtual mipmap. In Proceedings of SIGGRAPH ’98, pages 151–158, 1998.

[Wag04a] Wagner D. Terrain geomorphing in the vertex shader. Shader Programming Tips & Tricks with DirectX 9, 2004.

[Yoo05a] Yoon, S. Salomon, B. and Gayle, R. Quick-VDR: Out-of-core view-dependent rendering of gigantic models. IEEE Transactions on Visualization and Computer Graphics, 11(4):369–382, 2005.

Memory Efficient and Robust Software Implementation of the Raycast Algorithm

Aline A. de Pina
COPPE Sistemas/UFRJ,
aline@lcg.ufrj.br

Cristiana Bentes
DESC/UERJ,
cris@eng.uerj.br

Ricardo Farias
COPPE Sistemas/UFRJ,
rfarias@cos.ufrj.br

ABSTRACT

In this paper we propose two novel software implementations of the ray-casting volume rendering algorithm for irregular grids, called ME-Raycast (Memory Efficient Ray-casting) and EME-Raycast (Enhanced Memory Efficient Ray-Casting). Our algorithms improve previous work by Bunyk *et al* [1] in terms of complete handling of degenerate cases, memory consumption, and type of cell allowed in the grid (tetrahedral and/or hexahedral). The use of a more compact and non-redundant data structure, allowed us to achieve higher memory efficiency. Our results show consistent and significant gains in the memory usage of ME-Raycast and EME-Raycast when compared to Bunyk *et al* implementation. Furthermore, our results also show that handling of degenerate cases generates accurate images, correctly rendering all the pixels in the image, while Bunyk *et al* implementation fails in rendering up to 38 pixels in the final image. When we compare our algorithms to other robust rendering algorithm, like ZSweep [2], we have considerable performance gains and competitive memory consumption. We conclude that ME-Raycast and EME-Raycast are efficient methods for ray-casting that allow in-core rendering of large datasets with no image errors.

Keywords: Volume rendering, Ray-casting.

1 INTRODUCTION

Direct volume rendering has become a popular technique for visualizing volumetric data from sources such as scientific simulations, analytic functions, and medical scanners such as MRI, CT, and ultrasound. A big advantage of direct volume rendering is to allow the investigation of the interior of the data volume, because the objects are considered as composed of a semi-transparent material.

Volumetric data used in volume rendering is usually represented in the form of a regular or irregular grid. Regular grids are built with a rigid topological framework, and can be represented in an implicit form. Irregular grids, on the other hand, have the advantage of generality since they can conform to nearly any desired geometry, and thus, they are useful to represent complex geometries in a compact way.

Although several algorithms and methods have been proposed to efficiently render irregular grids, the most popular one is the **ray-casting** method. In this method, rays are casted from the viewpoint through every pixel of the image what determines which cells of the volume each ray intersects. Every pair of intersections is used to compute a contribution for the pixel color and opacity. The ray stops when it reaches full opacity or when it leaves the volume.

There are many different implementations of the ray-casting algorithm, [6, 5, 7, 8]. Only a few software solutions, however, deal with irregular grids. Garrity [3] proposed an efficient method for ray-casting irregular grids using the connectivity of cells. In his method, as the ray intersects one cell, it must exit through one of its faces. At this point it is only necessary to check intersections of the ray with the cell's faces. Therefore, Garrity used the connectivity of the data to move from cell to cell of the grid, in order to reduce the cost of identifying the cells which the ray intersects. This scheme leads to a quadratic cost on the number of cells. Later, Bunyk *et al* [1] improved Garrity's work by determining for each pixel an ordered list of intersections on external visible faces. This allows them to efficiently enumerate which boundary face intersects a given ray, and the correct order of the entry points for the ray. The rendering process follows Garrity's method, but when a ray exits the grid, the algorithm can easily determine in which cell the ray will re-enter the grid. This approach becomes simpler and more efficient than Garrity's propose, however it keeps some large auxiliary data structures.

The memory consumption of Bunyk *et al* approach is very high. This can have some implications in the algorithm efficiency when the computer does not have enough main memory. In addition, the amount of memory used by the ray-casting algorithm could complicate its implementation in the graphics hardware. Nowadays, this be-

 ISBN 978-80-86943-98-5

comes a huge obstacle for achieving real-time performance in rendering.

Besides the memory consumption problem, Bunyk *et al* approach has other shortcomings. First, there are some degenerate cases that cannot be handled by their algorithm. And second, it deals only with tetrahedral grids. In this work, we propose two novel ray-casting algorithms based on Bunyk *et al* approach, but improving it in different ways. Our goal is to develop memory efficient ray-casting algorithms that provide accurate results. Our approaches: (i) completely handle degenerate cases; (ii) use different data structures that are much smaller than the ones used in Bunyk *et al* approach; and (iii) deal with both tetrahedral and/or hexahedral grids.

Our algorithms, called ME-Raycast (Memory Efficient Ray-casting) and EME-Raycast (Enhanced Memory Efficient Ray-Casting), presented consistent and significant gains in memory usage over Bunyk *et al* approach. Our gains were not only in memory usage, but also in the correctness of the final image. Bunyk *et al* approach did not handle all possible degenerated cases, so it generated some incorrect pixels in the image.

We also compared our algorithms to other robust direct volume rendering algorithms based on cell projection paradigm, ZSweep. Our algorithms outperform ZSweep for all datasets. In terms of memory usage, for smaller images resolutions, ME-Ray spends more memory than ZSweep. EME-Ray, otherwise, spends less memory than ZSweep for most of the cases.

The remainder of this paper is organized as follows. In the next section we relate our work to others in the field of volume rendering of irregular grids. Section 3 describes our ray-casting algorithms and the improvements we made on Bunyk's approach, and shows how our algorithms handle the degenerate cases. In section 4 we present the results of our most important experiments. Finally, in section 6, we present our conclusions and proposals for future work.

2 RELATED WORK

There are mainly two categories of algorithms for direct volume rendering on irregular grids: ray-casting and projection.

Ray-casting algorithms are usually called image-space methods, since in its outer loop, it iterates over all the pixels of the output image. In the work by Garrity [3], as mentioned before, for each ray, exterior faces are tested to find the first intersection point. After that, the cells are traversed using the connectivity relation between them. This work was further improved by Bunyk *et al* [1], by computing for each pixel a list of intersections on external visible faces, and easily determining the correct order of the entry points for the ray. These two are all-software approaches, which means that they do not require any graphics hardware. Our work is also an all-software implementation, but provides improvements over Bunyk *et al* work. Weiler *et al* [12], on the other hand, implemented ray-casting using the graphics hardware. They find the initial ray entry point by rendering front faces, and then traverse through cells using the fragment program by storing the cells and connectivity graph in textures. Their method, however, work only on convex unstructured data, and is based on GPU programming.

Another class of rendering algorithms is the one that performs the render based on the sweeping paradigm to lower the cost of the ray-casting. The first work in this class was developed by Giertsen [4]. In his work, a plane sweeps the dataset in the up direction, or in the direction of Y axis, intersecting with cells. For every line of pixels of the image, all intersections of the sweeping plane with the cells of the grid is approximated by a regular 2D grid, and a bidimensional raycast is performed. One weakness of this method is the approximation imposed in the accommodation of the 2D grid, result of the intersection of the plane sweep with the data cells, onto the regular grid. Later, the work by Silva *et al* [10] improved Giertsen work. The Lazy Sweep algorithm avoids the approximation mentioned above.

Projection algorithms, on the other hand, reconstruct the image from the object space to the image space. The projection requires that the cells are first sorted in visibility ordering and then composed to generate their color and opacity in the final image. The first algorithm to be fully implemented to use projection was the ZSweep by Farias *et al.* [2]. The algorithm was implemented using only the CPU, what provided flexibility and easy parallelization. The ZSweep is a simple and efficient face projection rendering algorithm. ZSweep sweeps the dataset vertices, in depth order, with a plane perpendicular to the viewing direction. When the sweep plane hits a vertex, ZSweep project the faces incident on that vertex. To achieve memory efficiency, they used a mechanism called early ray composition. We used ZSweep algorithm as a baseline for our performance evaluation, in order to compare the speed and memory usage of our ray-casting algorithms over a projective one. The great advantage of projective methods is that they are efficiently implemented in programmable graphics hardware. Several cell projection algorithms were implemented using hardware graphics (e.g., [13], [9], [11]).

3 OUR APPROACHES

The main goal of ME-Raycast and EME-Raycast algorithms is to combine correctness of the results with effi-

 ISBN 978-80-86943-98-5

ciency in memory usage, without degrading the execution time.

For both algorithms, the traversal for each pixel starts in the same way proposed in Bunyk *et al* implementation. We project the visible faces on the screen and keep for each pixel the list of intersection points which enters the volume. Nevertheless, for the internal grid adjacency representation, ME-Raycast and EME-Raycast use completely different data structures. In fact, EME-Raycast was developed as an optimization of ME-Raycast in terms of memory usage. EME-Raycast uses simpler data structures than ME-Raycast. Our algorithms also include an identical and efficient method to deal with the degenerate cases that can occur during the ray traversal process, described in section 3.3.

3.1 ME-Raycast Algorithm

Before explaining the ME-Raycast algorithm itself, we describe its basic data structures. These data structures are also used by EME-Raycast, except for the most memory expensive array and some auxiliary structures, which are eliminated in EME-Raycast to lower memory consumption.

Basic Structures ME-Raycast keeps three basic structures: the vertex array (`Points_VEC`), the cell array (`Cells_VEC`) and the face array (`Faces_VEC`). There are also some auxiliary structures: the `Use_Set` of a vertex v is a list of all cells incident on v (see Farias *et al* [2]); the `Neighbor_Array` of a cell c is an array of indices of all neighboring cells of c; and the `Triangular_Faces` of a cell c is an array of indices of the triangular faces that bound c (in hexahedral cells, the faces need to be broken in two triangular faces).

The `Use_Set` array substitutes the `referredBy` list used in Bunyk's implementation. The `Use_Set` for each vertex, is a list of all cells incident on the vertex, in contrast with the `referredBy` list, which is a list of faces incident on the vertex. The `Use_Set` can be created in the preprocessing phase, in $O(c)$, where c is the number of cells. We allocate an array of integers (`int_array`), of the size of the number of vertices. For each cell, we loop through each of its vertices, and increment the element of `int_array` indexed by the number of the vertex, and a global counter. At the end, we know how many cells are incident on each vertex and the global total of incident cells on every vertex. Then, we allocate another array (`Use_Set`) using the global counter. We repeat the loop on the array of cells and fill in the `Use_Set` of each vertex.

Another step in the preprocessing phase is to find for each cell C_i its *face-neighbor* cells, which are the cells that share a face with C_i, and create the `Neighbor_Array` for C_i. We determine all the *face-neighbor* cells by scanning the `Use_Set` of the vertices of the cell. During this scanning, we create for each cell a list with the indices for its *face-neighbor* cells. We save a great amount of memory by keeping such lists on the cell structure instead of on the face structure (as done in Bunyk's method), since the number of faces is always greater than the number of cells. This information speeds up the process of stepping through the grid during ray-casting.

The `Faces_VEC` array is created on demand during the raycast process, as the faces are intersected by the rays. Only intersected faces are inserted. As a face is inserted, all its related parameters are computed. The number of faces in the array will depend on the image resolution and on the size of the dataset. For example, for a small resolution image and a large scale dataset, lots of faces will never be intersected by any ray and consequently will not be created by the process. Bunyk's method, on the other hand, inserts all faces in the preprocessing phase and compute their parameters at the beginning of the rendering. Processing time is saved, but with the cost of great memory overhead.

Algorithm The ME-Raycast algorithm can be divided into two phases: the preprocessing phase, and the core engine. Just like Bunyk's implementation, the preprocessing is performed while the dataset is read. In the preprocessing phase, the following steps are performed:

1. Read and store the vertices and cells of the dataset, creating `Points_VEC` and `Cells_VEC`.
2. Generate the `Use_Set` list for each vertex.
3. Determine for each cell its *face-neighbor* list.

To identify external faces of a cell, we store its own index, indicating that there is no *face-neighbor* cell sharing this face. We also create a list with all external faces. This list keeps, for each face, the index to the cell and to the relative face in the cell. The core engine of ME-Raycast algorithm performs the rendering process. For each point of view, ME-Raycast execute the following code.

```
Project external faces
   creating Ext_Faces;
For each pixel
  While( Ext_Faces not empty){
    Repeat {
      Find next intersection by
      checking other cell's faces;
      If (no intersection)
         check degenerate case;
      Accumulate colors/opacity;
```

 ISBN 978-80-86943-98-5

```
    } While (next intersected
             face is internal)
  }
```

The visible external faces (the ones whose normals have angles greater than 90° with the viewing direction) are projected on the screen, generating for each pixel a list of intersections with the external faces. These intersections will be used to start the raycast for each pixel.

To visualize the dataset from a different point of view, the list of faces is reused, saving processing time. Only the faces parameters must be recomputed for all faces. Also, our implementation allows both parallel and perspective projections.

3.2 EME-Raycast Algorithm

EME-Raycast and ME-Raycast have similar algorithms, but in EME-Raycast we have removed some data structures used in ME-Raycast.

The basic structures used in the EME-Ray are only two: the array of vertices (`Points_VEC`), and the array of cells (`Cells_VEC`). We have removed the array of faces (`Faces_VEC`), since it was one of the most memory expensive structures in ME-Raycast. Without the array of faces, we can save about $5*2*f$ bytes of memory, where f is the total number of triangular faces in the data set. Therefore, in a dataset with about 1 million faces, we are save about 52 Mb of memory.

From the cell structure, we removed the array `Triangular_Faces`. As tetrahedral cell uses 36 bytes, saving 16 bytes, and a hexahedral cell uses 60 bytes, saving 48 bytes of memory. In all we save about $(16*t+48*h)$ bytes of memory, where t is the total number of tetrahedral cells and h is the total number of hexahedral cells.

The vertex structure is identical to the one used in ME-Ray. However, the data structures removals are responsible for increasing the execution time. As we do not store the faces anymore, we need to recalculate the parameters for verification of ray intersection every time that a new face is checked.

3.3 Handling Degeneracies

The intersection between a ray and a face of a cell is the result of the algebraic calculation of the intersection between a line and a plane, see [1]. For tetrahedral grids, it is the intersection between the line defined by the ray path and the plane defined by the three vertices of a triangular face. For hexahedral grids, where each quadrangular face is defined by four vertices, the intersection is found by splitting the quadrangular face into two triangular faces, and performing the same calculation mentioned above, for each face. This way, we do not need to worry about the four points being coplanar to define a plane.

Bunyk *et al* use the Point-Within-Triangle algorithm to determine the ray intersections and to look for the next cell the ray will intersect. In their algorithm, however, degenerated situations may arise when the ray hits a vertex or an edge. In Figure 1, we exemplify in 2D the case where the ray hits a vertex. The blue cell corresponds to the first cell that the ray intersects, called the current cell. Bunyk *et al* approach would check only the cells neighboring the current cell faces, i.e., faces of the cells A and E. However, the ray does not intersect with neither A nor E faces. In this case, the final color of the pixel will be wrong, since the composition process will be interrupted.

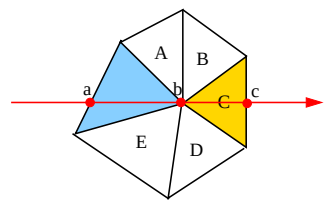


Figure 1: 2D example where the ray hits a vertex and Bunyk approach does not find c

To avoid this type of error, we propose a different kind of verification to look for the next cell intersected by the ray. The idea is to allow the continuation of the ray traversal, by looking for the next cell scanning the `Use_Set` of each vertex which determine the current cell. In the example in Figure 1, this scanning will return cells A, B, C, D and E. Therefore, this scheme asserts that another intersection will be found in a face of cell C, guaranteeing that the ray traversal will continue and the image will be correctly generated.

When the ray hits an edge, the problem can be solved by the same procedure explained above. In Figure 2, we show an 3D example for this case. In this example, the blue cell of (a) corresponds to the first cell that the ray intersects. The next intersection is in the edge V_0V_1 in the point b. Bunyk *et al* approach would look for the next intersection in all the faces adjacent to V_0V_1 edge. These faces are shown in (b). As we can observe in the figure, it is not possible to find the next intersection in the adjacent faces of V_0V_1. Our approach, on the other hand, uses the `Use_Set` of the vertices of the blue cell to find the next intersection. Using the `Use_Set` of V_0 and V_1, we find the yellow cell of (a). In the yellow cell, we find the next face the ray intersects, determining c.

4 EXPERIMENTAL RESULTS

In this section we evaluate the performance and memory usage of ME-Raycast (**ME-Ray**) and EME-Raycast

 ISBN 978-80-86943-98-5

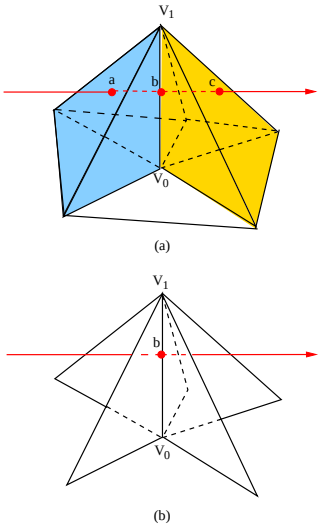


Figure 2: 3D example where the ray hits an edge and Bunyk approach does not find *c*

(**EME-Ray**). Our evaluation uses two different baselines for the comparisons. The baselines are two direct volume rendering algorithms for irregular grids: the Bunyk *et al* implementation of ray-casting (**BUNYK**); and the ZSweep cell projection algorithm (**ZSweep**). The ideas behind these comparisons are to: (1) Measure the improvements over Bunyk work in terms of memory usage and correctness of the final image. (2) Put our results in perspective, with respect to other robust rendering algorithm (that generates correct final images). Following, we briefly describe the two baselines, show the datasets used in our experiments, and, then, describe our performance analysis.

4.1 Baselines

BUNYK: Bunyk *et al.* implementation initially projects the external visible faces on the screen, creating for each pixel, a list of the intersections generated by these projections. The process starts by projecting all faces whose normal make an angle greater then 90° with the viewing direction. After projecting every visible face, the algorithm knows, for each pixel, which face through which the ray enters the volume. Since the algorithm computes all cell's neighbors in preprocessing, it is computed, in constant time, the next face the ray is going to intersect. For every two consective intersections, opacity and color integrations are computed. Once there is no more entry point for the pixel, the ray has left the volume, and the process is finshed for the current pixel. It is important to notice that, the list of faces created to carry on this method is responsible for about half the memory usage of this implementation.

Table 1: Datasets used in our experiments.

Dataset Information				
Datasets	Vertices	Faces	Boundary	Cells
Blunt Fin	40.960	381.548	13.516	187.395
Comb. Chamber	47.025	437.888	15.616	215.040
Oxygen Post	109.744	1.040.588	27.676	513.375
SPX	149.224	1.677.888	44.160	827.904
Delta Wing	211.680	2.032.084	41.468	1.005.675
Hexa	2.684	6.432	1.344	1.920

ZSweep: The ZSweep algorithm is a direct volume rendering algorithm based on the sweeping paradigm, and built over the success of prior sweep approaches [10]. The main idea of ZSweep algorithm is the sweeping of the data with a plane parallel to the viewing plane XY, towards the positive z direction. The sweeping process is performed by ordering the vertices by their increasing z coordinate values, using a heap sort, and then retrieving one by one from this data structure. For each vertex swept by the plane sweep, the algorithm projects, onto the screen, all faces that are incident to it. When a face is projected onto a given pixel, the result is equivalent to the intersection of the ray emanating from this same pixel and the face being projected. ZSweep stores its z-value, and other auxiliary information, in sorted order in a list of intersections for the given pixel, called pixel list. To achieve memory efficiency, ZSweep uses a mechanism called early composition. The composition of the intersections in a pixel list is performed as the *target-Z* is reached. The *target-Z* represents the maximum z coordinate among the vertices adjacent to the first vertex encountered by the sweeping plane. When the plane reaches a target z, the next target z will be again the maximum z coordinate among the vertices adjacent to the current reached target, and the process continues.

4.2 Workload

Our experiments were conducted in a Pentium 4, 2.80GHz with 1GB of memory, running Linux Fedora Core 2. We have used five different tetrahedral datasets: Blunt Fin, Combustion Chamber, Oxygen Post, SPX and Delta Wing, and also one small hexahedral dataset, called Hexa, used only to show our handling of hexahedral grids. The number of vertices, faces, boundary faces and cells for each dataset are listed in Table 1. We also varied the image sizes, from 128×128 to 1024×1024 pixels.

4.3 ME-Raycast and EME-Raycast Performance

In this section we evaluate ME-Raycast and EME-Raycast algorithms, compared to ZSweep and BUNYK results. In terms of the number of pixels rendered, ME-Ray, EME-Ray and ZSweep rendered all the pixels,

 ISBN 978-80-86943-98-5

generating a correct image. However, since BUNYK does not handle all possible degenerate cases, it fails in rendering the amount of pixels shown in Table 2. As we can observe in this table, for Blunt Fin, Oxygen, Delta and SPX, BUNYK generates a great amount of flaws in the 512×512 and 1024×1024 images. Up to 38 pixels were not rendered correctly.

Table 2: Pixels not rendered by BUNYK

Bad pixels - BUNYK					
Image Size	Blunt Fin	Combustion	Oxygen	Delta	SPX
128^2	2	-	-	1	-
256^2	3	1	4	2	3
512^2	10	-	11	7	5
1024^2	31	2	38	17	18

Tables 3, 4, 5, 6 and 7 show the results of time and the amount of memory consumed for Blunt Fin, Combustion Chamber, Oxygen Post and SPX datasets, respectively, rendered by ME-Ray and EME-Ray when compared to BUNYK and ZSweep execution, for four different image resolutions, 128×128, 256×256, 512×512, and 1024×1024. The results for the Hexa dataset are not considered because it is a very small dataset, and BUNYK algorithm cannot handle heaxahedral datasets. The percentages presented in these tables correspond to the ratio of our algorithm (ME-Ray or EME-Ray) result over the baseline (BUNYK or ZSweep). In other words, we consider BUNYK or ZSweep results as 100% and are presenting how much we increase or decrease this baseline.

Table 3 presents the results for the Blunt Fin dataset. Comparing ME-Ray and EME-Ray with BUNYK, we observe that they use considerably less memory than BUNYK. ME-Ray uses, for a 1024×1024 image, almost the same memory BUNYK uses for a 128×128 image. EME-Ray uses 3.5 times less memory than BUNYK for a 512×512 image and 2.5 times less memory for a 1024×1024 image. These significant reductions in memory usage comes with an increase in the execution time. The increase, however, is only about 26% for ME-Ray for a 1024×1024 image. When compared to ZSweep, we observe that EME-Ray outperforms ZSweep in terms of render time and memory usage for the three larger image precisions.

Table 4 shows the results for Combustion Chamber dataset. ME-Ray and EME-Ray also consume less memory than BUNYK. For a 512×512 image, EME-Ray spends 3.6 times less memory than BUNYK and, for a 1024×1024 image EME-Ray spends 2.7 times less memory. In terms of execution time, BUNYK outperforms ME-Ray, but for a 1024×1024, ME-Ray is only 6% slower. When compared to ZSweep, ME-Ray spends more memory, but is faster, and EME-Ray is faster and uses less memory for larger images.

Table 3: Blunt Fin Data Results

Image		Time		Memory	
		BUNYK	ZSweep	BUNYK	ZSweep
128^2	ME-Ray	145%	32%	52%	229%
	EME-Ray	225%	50%	24%	106%
256^2	ME-Ray	133%	30%	61%	184%
	EME-Ray	281%	64%	25%	75%
512^2	ME-Ray	138%	32%	69%	96%
	EME-Ray	335%	78%	28%	39%
1024^2	ME-Ray	126%	29%	75%	40%
	EME-Ray	304%	70%	39%	20%

Table 4: Combustion Chamber Results

Image		Time		Memory	
		BUNYK	ZSweep	BUNYK	ZSweep
128^2	ME-Ray	184%	57%	73%	351%
	EME-Ray	166%	51%	24%	115%
256^2	ME-Ray	141%	43%	74%	287%
	EME-Ray	215%	65%	25%	96%
512^2	ME-Ray	153%	57%	75%	169%
	EME-Ray	232%	87%	28%	63%
1024^2	ME-Ray	106%	38%	76%	76%
	EME-Ray	224%	80%	37%	37%

Table 5 shows the results for Liquid Oxygen Post dataset. ME-Ray and EME-Ray use considerably less memory than BUNYK. For larger images, BUNYK uses about 3 times more memory than our algorithms. As the image size grows, however, BUNYK becomes much more faster than EME-Ray. Compared to ZSweep, ME-Ray is about 2.6 times faster and uses 1.5 times less memory for the largest image precision. For a 512×512 image, EME-Ray is about 1.5 times faster and uses 1.6 times less memory than ZSweep.

Table 5: Liquid Oxygen Post Results

Image		Time		Memory	
		BUNYK	ZSweep	BUNYK	ZSweep
128^2	ME-Ray	162%	24%	44%	216%
	EME-Ray	200%	29%	24%	118%
256^2	ME-Ray	137%	39%	50%	205%
	EME-Ray	268%	76%	24%	100%
512^2	ME-Ray	142%	41%	57%	142%
	EME-Ray	276%	79%	26%	64%
1024^2	ME-Ray	136%	38%	66%	70%
	EME-Ray	289%	82%	30%	32%

Table 6 shows the results for the largest dataset, Delta Wing, that has more than 1 million cells. ME-Ray method is about 1.5 times slower than BUNYK, but uses 1.6 times less memory and EME-Ray uses less than 30% of the memory used by BUNYK. Compared to ZSweep, ME-Ray is faster for all images sizes, but consumes more memory. For small images, even EME-Ray consumes more memory than ZSweep. This is due to the indices we keep for the neighboring cells for each cell. Nevertheless, EME-Ray performs significantly better than ZSweep.

 ISBN 978-80-86943-98-5

Table 6: Delta Wing Results

Image		Time		Memory	
		BUNYK	ZSweep	BUNYK	ZSweep
128^2	ME-Ray	141%	15%	38%	200%
	EME-Ray	150%	16%	23%	124%
256^2	ME-Ray	162%	26%	44%	216%
	EME-Ray	203%	32%	24%	118%
512^2	ME-Ray	139%	32%	51%	209%
	EME-Ray	244%	56%	24%	100%
1024^2	ME-Ray	148%	33%	61%	152%
	EME-Ray	271%	61%	27%	68%

Table 7 shows the results for the SPX dataset. Although ME-Ray is slower than BUNYK, it uses less memory. EME-Ray uses even less memory. For example, to create a 512×512 image, EME-Ray uses about 1/4 of the memory necessary for BUNYK to create a 256×256 image. Compared to ZSweep, ME-Ray and EME-Ray are faster than ZSweep. In terms of memory usage, ME-Ray uses more memory than ZSweep and EME-Ray uses 1.2 times less memory than ZSweep for a 1024×1024 image.

Table 7: SPX Results

Image		Time		Memory	
		BUNYK	ZSweep	BUNYK	ZSweep
128^2	ME-Ray	120%	20%	51%	273%
	EME-Ray	197%	33%	23%	125%
256^2	ME-Ray	154%	41%	68%	349%
	EME-Ray	178%	48%	24%	121%
512^2	ME-Ray	136%	46%	72%	320%
	EME-Ray	192%	66%	24%	109%
1024^2	ME-Ray	116%	41%	74%	222%
	EME-Ray	222%	79%	27%	83%

5 DISCUSSION

ME-Ray and EME-Ray had obtained consistent and significant gains in memory usage over BUNYK. In terms of the image resolution, we can observe that the gains of ME-Ray and EME-Ray over BUNYK are bigger for smaller image sizes. This occurs because BUNYK creates at once an array with all the faces in the dataset, and this does not depend on the image size. While ME-Ray creates the faces as they are intersected by the rays. Otherwise, in terms of the dataset size, as we expected, when the dataset increases, the reductions in memory usage of ME-Ray also increases. This result confirms that our data structures are set to handle big datasets.

Furthermore, the reductions in memory requirements we obtained with our data structures, allowed us to use double precision in the parameters to calculate the intersection between a ray and a face. BUNYK uses float for these parameters, consequently causing some precision errors. We have made some experiments with BUNYK algorithm increasing the parameters precision to double and obtained better images. On the other hand, the memory requirements increased about 12F bytes (where F is the number of faces in the dataset).

The increase in the execution time, when compared to BUNYK, comes from the fact that we have to scan through out the `Use_Set` of the vertices to perform the ray traversal. Since the `Use_Set` keeps the indices for the cells incident on each vertex, its likely to occur double intersection computation for internal faces. On the other hand, BUNYK keeps all faces incident on the vertices which makes it faster to compute such intersections, while spending more memory. In our experiments, however, we are only comparing executions where the whole dataset fits in main memory for both methods. As the memory usage increases, the rendering will need to use of the virtual memory mechanisms of the operating system, which would have great influence on the overall execution time.

It is also important to notice that ME-Ray and EME-Ray gains over BUNYK are not only in memory usage, but also in the correctness of the final image. BUNYK does not handle all possible degenerated cases. For 1024×1024 images, in all the datasets, BUNYK algorithm generates some flaws in the image. For Delta Wing, for example, BUNYK fails in rendering 17 pixels. This causes some black spots in the image as we can observe in the 512×512 image of Figure 3.

When compared to ZSweep, ME-Ray outperforms ZSweep, in execution time, significantly for all the datasets and all the image resolutions. Although it is not an intuitive result, it is explained by the fact that, in ZSweep, while the *target-Z* is not reached, the pixel list increases. The bigger the list is, the more expensive is the insertion, since it is ordered. Depending on the dataset, the *target-Z* could be a bad parameter to start the composition. ME-Ray, on the other hand, composes the pixels on-the-fly as each intersection is found.

Another important difference in the performance of ME-Ray and EME-Ray, when compared to ZSweep, is the dataset structure. More "irregular" datasets with holes and much more external visible faces would benefit ZSweep, since ME-Ray and EME-Ray would have to compute more external faces intersection. This, however, is not the case for our workload, except for SPX, that provides the smaller performance difference between ME-Ray and EME-Ray compared to ZSweep.

In terms of memory requirements, ME-Ray spends more memory than ZSweep, except for 1024×1024 image resolutions. EME-Ray, on the other side, spends less memory than ZSweep for most of the datasets and image resolutions. ZSweep increases linearly the memory requirement with the increase in the image size, since as the

 ISBN 978-80-86943-98-5

image size increases, each face projected will insert intersection units into more pixel lists. ME-Ray also increases linearly the memory requirements with the increase in the image size. This increase is due to the increase in the size of `Faces_VEC`, since more faces are intersected. EME-Ray, otherwise, does not have `Faces_VEC` data structure, so the memory usage maintains almost constant, even when the image size increases.

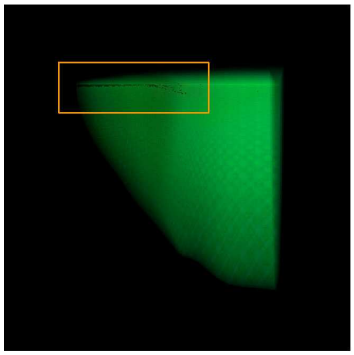


Figure 3: Delta (512x512) generated by BUNYK with bad rendered pixels highlighted by the orange box.

6 CONCLUSIONS

We proposed two novel ray-casting algorithms, ME-Raycast (Memory Efficient Ray-casting) and EME-Raycast (Enhanced Memory Efficient Ray-Casting). Our algorithms improve previous work by Bunyk *et al* in terms of memory consumption, type of cell allowed in the grid (tetrahedral and hexahedral), and complete handling of degenerate cases. Our goal in improving Bunyk *et al* work was to provide a software implementation of ray-casting that is memory efficient without performance degradation, and robust, i.e., generates correct images

Our experimental results showed that ME-Raycast and EME-Raycast are comparable in performance to Bunyk *et al* in most of the cases, but had obtained consistent and significant gains in memory usage over their approach. These results confirm that our data structures store only essential information. When compared to other accurate rendering algorithm, ZSweep, ME-Raycast and EME-Raycast obtained considerable performance gains, and competitive memory consumption. EME-Raycast by itself spends less memory than ZSweep for most of the datasets and image resolutions.

Our results also showed that ME-Raycast and EME-Raycast complete handling of degenerate cases generates accurate images, rendering correctly all the pixels of the image. Nevertheless, Bunyk *et al* work failed on rendering some pixels in the final image, generating incomplete results. Besides the memory and performance results, we showed that we can deal with grids represented by tetrahedra, hexahedra or both. As far as we know, they are the first ray-casting implementations which handle, at the same time, both types of irregular grids.

We conclude that ME-Raycast and EME-Raycast are efficient algorithms for ray-casting that allows the in-core rendering of big datasets, avoiding paging operations on disk. The low memory usage of our algorithms also makes them suitable for hardware-based implementations, in order to achieve real-time rendering. As future work, we consider the study of out-of-core versions of the codes that run on clusters of PCs.

REFERENCES

[1] P. Bunyk, A. Kaufman, and C. Silva. Simple, fast, and robust ray casting of irregular grids. *Advances in Volume Visualization, ACM SIGGRAPH*, 1(24), July 1998.

[2] R. Farias, J. Mitchell, and C. Silva. Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In *2000 Volume Visualization Symposium*, pages 91 – 99, October 2000.

[3] M. P. Garrity. Raytracing irregular volume data. In *VVS '90: Proceedings of the 1990 workshop on Volume visualization*, pages 35–40. ACM Press, 1990.

[4] C. Giertsen. Volume visualization of sparse irregular meshes. *IEEE Comput. Graph. Appl.*, 12(2):40–48, 1992.

[5] L. Hong and A. Kaufman. Accelerated ray-casting for curvilinear volumes. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 247–253, 1998.

[6] W. M. Hsu. Segmented ray casting for data parallel volume rendering. In *PRS '93: Proceedings of the 1993 symposium on Parallel rendering*, pages 7–14, 1993.

[7] K. Koyamada. Fast ray-casting for irregular volumes. In *ISHPC '00: Proceedings of the Third International Symposium on High Performance Computing*, pages 557–572, 2000.

[8] A. Neubauer, L. Mroz, H. Hauser, and R. Wegenkittl. Cell-based first-hit ray casting. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 77–ff, 2002.

[9] S. Rottger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 109–116, 2000.

[10] C. T. Silva and J. S. B. Mitchell. The lazy sweep ray casting algorithm for rendering irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):142–157, 1997.

[11] C. T. Silva, J. S. B. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 87–94, 1998.

[12] M. Weiler, M. Kraus, and T. Ertl. Hardware-based view-independent cell projection. In *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 13–22, 2002.

[13] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno. Tetrahedral projection using vertex shaders. In *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 7–12, 2002.

 ISBN 978-80-86943-98-5

Inverse Modeling and Animation of Growing Single-stemmed Trees at Interactive Rates

Steffen Rudnick* Lars Linsen† E. Gregory McPherson‡

* Department of Mathematics and Computer Science
Ernst-Moritz-Arndt-Universität Greifswald, Germany

† School of Engineering and Science
International University Bremen,§ Germany

‡ USDA Forest Service, Pacific Southwest Research Station
Center for Urban Forest Research, Davis, California

ABSTRACT

For city planning purposes, animations of growing trees of several species can be used to deduce which species may best fit a particular environment. The models used for the animation must conform to real measured data. We present an approach for inverse modeling to fit global growth parameters. The model comprises local production rules, which are iteratively and simultaneously applied to build a fractal branching structure, and incorporates the propensity of trees to grow towards light. The parameters of the local production rules are derived from global functions that describe the measured tree growth data over time. The production rules are influenced by the global light distribution, which is represented by the amount of light available at each position within the tree's crown. Since we want to allow the user to explore the tree's appearance interactively at any time during the animation, all modeling computations must be within a time frame that allows for interactive rendering rates. To this end, we developed a fast approximate algorithm for computing the light distribution. The rendering itself must also be fast; therefore, we sought a well-balanced compromise between photo-realism and performance. Because shadow computations play a key role for photo-realism, we developed a fast approximate shadow computation algorithm including soft shadows and self-shadowing. We applied our methods in order to model and animate the growth of seven single-stemmed tree species in an interactive setting.

Keywords
Tree Growth Modeling, Animation, Real Time Rendering.

1 Introduction

Trees produce benefits that enhance quality of life for city residents. Benefits include energy savings, air pollutant uptake, CO_2 sequestration, storm-water runoff reduction, increased property values and increased vitality in commercial areas. As trees grow larger, benefits increase as leaf surface area increases. However, the benefits can be offset when tree branches, and leaves conflict with other urban infrastructure, such as buildings, awnings, signs, traffic signals, and lighting. In dense urban environments limited space is a major constraint to tree planting. One goal of urban greening is to maximize functional benefits and minimize conflicts between trees and other infrastructure. To achieve this goal it is important to know (1) the mature size of different tree species, since both functionality and conflicts depend on tree size relative to the space that is available, and (2) the rate of growth for different trees, because this influences the length of time before conflicts occur, as well as how soon functional benefits are realized. Currently, landscape architects, city planners, and urban foresters lack tools that help them visualize how trees will grow over time, so that they can compare different species and select ones best suited to the site and their design objectives. Because trees are expensive to plant and can live to be hundreds of years old, computer-animated tree-growth modeling can help users make judicious choices that will pay large dividends over the long-term.

In computer graphics, several methods exist to describe and model computer-generated trees. The goal of these methods is to generate photo-realistic images of trees of selected species. The trees should appear as natural as possible and vary in appearance as they do

*srudnick@uni-greifswald.de
†l.linsen@iu-bremen.de
‡egmcpherson@ucdavis.edu
§Jacobs University Bremen as of Spring 2007

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright UNION Agency - Science Press, Plzen, Czech Republic.

in nature. Typically, the quality of the images, i.e. how natural the appearance of the trees is, is not quantitatively evaluated but just estimated by the human eye. Real measured data such as tree dimensions are not used to compare the models with nature.

For animations of trees growing over time, the complex growing process under the influence of many biological phenomena has to be considered. Some elaborate approaches exist. Our approach differs from most of these in two main ways:

1) Our tree growth is based on measured data.
2) Our entire tree growth modeling process is done on-the-fly, i.e. while users watch and interact with the animation. We achieve interactive rates for modeling and rendering.

The properties of our model allow users to interactively compare the growth of different tree species side by side. Because our model is based on real data the users confidence in the approach is increased. For a case study, we used global functions for seven single-stemmed tree species. Our growth model described in Section 3 is based on basic biological background on tree growth so that a realistic branching structure can be generated. We also used measured data for fitting the global shape properties of the respective species. Typically, a computer model for trees consists of a branching structure with position, length and orientation information for every branch. Local production rules are applied to each branch iteratively to generate the complex branching structure. We derive local production rules for our model based on the global functions given. Moreover tree growth cannot be determined by local rules only, but global phenomena like light seeking or shadow avoidance also have to be included. Thus our production rules also consider available light and its distribution.

For tree rendering, several photo-realistic approaches exist including sophisticated techniques such as the creation of soft shadows by multiple subsurface scattering. However, these approaches do not meet real-time requirements especially when dealing with steadily changing geometry. Thus we had to find a compromise between photo-realistic rendering and fast rendering. We observed that shadows are of high importance in this context. We developed a simple and fast method to approximate and render soft shadows. In addition, the effect of self-shadowing is included by drawing leaves with a luminance depending on the light distribution. Details of our rendering methods are described in Section 4.

2 Related Work

The modeling, simulation, and rendering of trees and plants is a well-studied topic in computer graphics. The formal description of trees is usually based on local production rules. Starting with a trunk and iteratively applying local production rules generates a complex branching structure.

The Lindenmayer systems (L-systems), introduced by the theoretical biologist Aristid Lindenmayer [Lin68], are the most common approach for a formal description of plants. In their book *The Algorithmic Beauty of Plants* [PL90], Prusinkiewicz and Lindenmayer describes the general concept of L-systems and their application to modeling different plants with different structures. They also introduced the concept of parametric L-systems in which the local production rules depend on parameters that are locally stored and updated. This concept has been used and extended in several algorithms [AK85, Blo85, LD99]. A survey of existing L-system approaches is given in [PHMH95].

Natural phenomena such as growth, death, reproduction, and information flow in growing plants can be modeled via L-systems. Prusinkiewicz et al. [PHHM97] explained how the L-system model applies to nature. The influence of the environment on the growth of plants is also considered.

For computer graphics applications, the main objective of modeling plants is to generate a highly realistic scene. Therefore stochastic tree models have been introduced to simulate the variety within one species. The individual, realistic-looking plants differ from each other and can be organized to render forests or fields [CSHD03] and even entire ecosystems [DCSD02]. The goal of the previous modeling systems differs from ours in that they weren't limited to interactive frame rates.

Approaches using inverse modeling of trees have been developed by Galbraith et al. [GMW04] and Prusinkiewicz et al. [PMKL01]. They used relative positions inside the crown to describe the local properties of the plant organs to generate trees that fit a given shape. Inverse modeling in this sense means, for example, that the relative position of a branch along the trunk (height of the starting point of the branch divided by the crown height) and the knowledge of the crown shape and dimension determine the length of the branch. This approach allows the use of real measured tree data such as height or crown shape.

Linsen et al. [LKMH05] presented a visualization method for tree growth at interactive frame rates using global functions to model the shape and size of the trees. They used some global parameters to determine the local production rules but their approach was too simple to produce realistic branch lengths. In their method, several biological phenomena were not considered including the fact that growth is periodical in years, or that plants try to avoid growing leaves in shadows.

Several approaches account for the light available for the growth of plants [Ben96, HB03, HdFBR04, SSBD03]. In these approaches a light source is modeled and the amount of light a leaf or a branch receives is computed. The illumination computation is very intense due to the complex and changing geometry of growing trees. The computations of the precise illumination inside a tree crown are further complicated by the reflected and transmitted light. To allow for in-

 ISBN 978-80-86943-98-5

teractive modeling, our light computation needs to be much faster. We propose a fast approximate computation of the light distribution that is sufficiently precise for growth modeling and for realistic shadow computations.

3 Modeling

3.1 Global Functions

Our tree growth model is based on global functions of time that describe global properties of the tree during growth. For modeling we use functions for the diameter at breast height f_{DBH}, tree height f_H, crown diameter f_{CD} and the crown height f_{CH}. These functions are derived from real measured data for seven single-stemmed tree species, the London planetree (*Platanus x acerifolia*), silver maple (*Acer saccharinum*), Modesto ash (*Fraxinus excelsior*), goldenrain tree (*Koelreutaria paniculata*), Southern magnolia (*Magnolia grandiflora*), Chinese pistache (*Pistacia chinensis*), and hackberry (*Celtis occidentalis*).

3.2 Biological Background

We briefly summarize the biological characteristics of tree growth for single-stemmed trees that are important to our approach; for a more detailed explanation, we refer the reader to the literature [SS02, Mat91, Nul68].

A tree is a plant that grows over years, building up a large branching structure. The growth of a branch segment is divided into two phases, primary growth and secondary growth. Primary growth is the elongation of a bud from last year's growth to a shoot terminated by an apical bud. During primary growth, leaves and lateral buds are produced. After the first year's growth, branch segments start the secondary growth phase, which is characterized by steady thickening while elongation stops. The apical bud always produces the longest shoot and elongates the branch, while the lateral buds produce shorter shoots, a phenomenon known as apical dominance. The lateral buds do not always produce a shoot. The probability that they will produce a shoot is mainly dependent on the available light. A schematic example of the growth of one branch is shown in Figure 1.

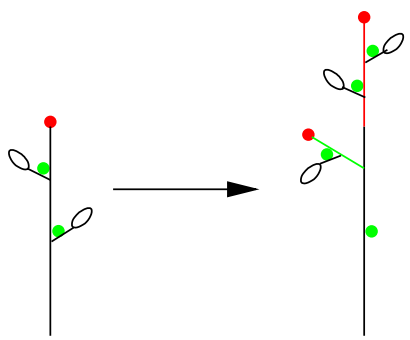


Figure 1: Annual growth of a branch. The apical bud (red) on the left produces an apical shoot (red) on the right with a new apical bud (red) and lateral buds (green). One of the lateral buds on the left produces a lateral shoot (green).

The leaves growing along the shoots produce energy by the process of photosynthesis. Thus trees tend to grow leaves in positions with good light. The different responses of plants to light and shadow are explained and discussed in *Photomorphogenesis in Plants* by Kendrick and Kronenberg [KK86]. Branches receiving enough light try to produce many leaves to produce energy. Thus they tend to bifurcate more often. Branches exposed to less light do not bifurcate as much; they tend to grow faster and to produce longer branches in order to reach for more light.

3.3 Tree Growth

A branching structure consists of bifurcations and branches. To generate complex branching structures, computer models typically use iteratively and simultaneously applied local production rules. In order to generate branching structures that consider global properties like the global functions derived in Section 3.1 or the available light distribution, we have to extend these local production rules to rules that couple local reproducibility with global growth properties.

In our model, a branch consists of branch segments representing the annual growth. A branch segment is defined by its length, diameter, starting point, and direction. A bifurcation is characterized by the bifurcation angle, the divergence angle, and the ratios of length and diameter between the parent branch and the child branches. The bifurcation angle describes the angle between the new and old branches and the divergence or twisting angle describes the change in orientation because not all branches lie in one plane. In addition to branches and bifurcations, we also model leaf growth. Flowers and fruits can be handled in the same way but are not included in our model, since our goal is the visualization of growth over several years while seasonal changes are omitted. The leaves spiral around the branch equidistantly. The leaves only grow on young and small branches that are exposed to sufficient light.

Tree growth is modeled in discrete steps each representing $\frac{1}{n}$-th of a year where n is exchangeable and can accommodate existing graphics hardware equipment. In our implementation we use $n = 20$. In each step, the production rules are applied to all branch segments. Using a pseudocode description, the recursive function for growing branches is given by:

```
grow (branch)
   if (primary growth)
        compute new length
   if (end of primary growth)
        produce apical branch segment
```

 ISBN 978-80-86943-98-5

```
        produce lateral branch segment
    if (secondary growth)
        compute radius
        grow(apical branch)
        if (lateral branch exists)
            grow(lateral branch)
```

The result of "produce lateral branch segment" is a new lateral branch segment but only if the global parameters allow its generation. Otherwise nothing is done. A branch segment is in the phase of primary growth for one year. Afterwards, the secondary growth starts.

Every branch has an order representing its depth in the branching structure (Weibull ordering). The trunk has order 0, the branches connected to the trunk have order 1 and so on.

3.3.1 Trunk length

The trunk consists of two parts, one below the crown and one surrounded by the crown. The lengths of both are controlled by the global functions f_H and f_{CH}. The length of the part below the crown is calculated as the difference between the measured tree height and the measured height of the crown $l(t) = f_H(t) - f_{CH}(t)$ where t is the age of the tree in years. The length of the part surrounded by the crown is given by the measured height of the crown, $f_{CH}(t)$. The whole trunk consists of many trunk segments each representing annual growth and the last one is the only one in the primary growth phase. Thus the growth of the last segment is given by the difference between old and new crown height, $f_{CH}(t) - f_{CH}(t - \Delta t)$ for time steps of size Δt.

3.3.2 Branch length

For calculating the length of each branch, its order, its relative position, $p_r \in [0,1]$, along the parent branch, and the length of the parent branch, l_p, are used. The length calculation of the branches is done with the help of a crown shape function, $c : [0,1] \rightarrow \mathbb{R}$, which is represented by a geometric function that returns the relative length of a branch at a relative position, p_r, along the trunk to fit the crown shape (see Figure 2). The crown shape function is applied recursively for the entire branching structure, i.e., the length of each branch is determined with respect to the crown shape function applied to its parent branch.

Multiplying the relative length of a branch, $c(p_r)$, by the length of the parent branch, l_p, by an order-depending factor, and by a length scaling factor, s_l, gives the new length

$$l \quad = \quad s_l \cdot \frac{c(p_r) \cdot l_p}{0.5 + 0.5 \cdot order} \; .$$

This scaling factor $s_l \in \mathbb{R}_+$ randomly assigned to each branch during its initialization is uniformly distributed over $[0.6, 1]$. To allow for a few branches to break out

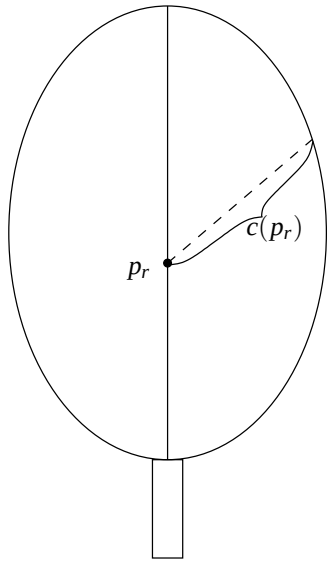


Figure 2: Example for elliptical crown shape function: p_r is the relative position of the starting point of a branch and $c(p_r)$ the computed relative branch length.

of the given shape and grow towards empty regions, 5% of the branches of order ≥ 2 are assigned a factor uniformly distributed over $[1.2, 1.5]$. Again the growth is only added to the length of the last branch segment, i.e., the only segment of the branch in primary growth phase.

Since branches grow towards light, we adjust our model so that light influences the scaling factor s_l. When there is only a little light at the branch apex, the scaling factor increases such that the branch reaches out of the shadow. When there is plenty of light at the apex of the branch, the scaling factor does not increase.

3.3.3 Radius

The radius of each branch segment is computed using the diameter at breast height function f_{DBH}. Similar to the method of scaling the length with factor s_l, we apply a scaling factor $s_r \in [0,1]$ to the radius. The new radius for each branch segment or trunk segment is given by

$$r(t) = s_r \cdot f_{DBH}(t) \; ,$$

where t, again, denotes the age of the branch segment in years. The scaling factor s_r for a branch segment is computed during initialization depending on the scaling factor $s_{r,f}$ of the previous branch segment as

$$s_r = s_{r,f} \cdot (1 - s_{r,f} \cdot 0.7) \; .$$

3.3.4 Bifurcation

Once the primary growth of a branch segment stops, the production rules for new branches have to be applied. As mentioned in Section 3.2, the probability

 ISBN 978-80-86943-98-5

that a bud near the apex will produce a shoot is much larger than for the other buds. In our model, the apical bud always produces a new shoot, which elongates the branch and becomes a new segment. One lateral bud can produce a new shoot which starts a new branch of one order higher. The probability is determined by the local tree density, the available light at this position, and a random factor. Basically, plenty of light leads to many bifurcations and the absence of light leads to no further bifurcations.

For these new shoots, the scaling factors s_l and s_r are generated, and the angles characterizing the bifurcation are determined. The angles used are inspired by the ones that can be seen in nature in the respective tree species. In particular, branches tend to spiral around the parent branch.

3.4 Light

For the purpose of interactive modeling and visualization light calculations must be fast. We use a method to approximate the available light at each position inside the crown of the tree. We divide an axes-aligned box that covers the full-grown tree into discrete volumes. We send light rays simulating daylight through the scene that lose intensity as they pass through the cells. The loss of intensity is proportional to the percentage of the cell's space covered by branches and leaves.

After each growth step, the light distribution within the crown has to be recomputed. Therefore, we define equidistant planes parallel to the ground in the crown space. Thus, the bounding box of the tree is partitioned into slabs. Each branch is projected into the lower plane in each slab using parallel projection while information on its radius and number of leaves is stored.

Each plane is divided into squares forming a regular grid. The size of the grid is adapted to the size of the crown. For each square S, a density, $D \in [0,1]$, is computed that measures the percentage of the square not covered by branches or leaves

$$D = \prod_{i \in S} (1 - A(i)) \cdot (1 - l(i))$$

where $A(i)$ is the area of the part of the projected branch i that intersects S, and $l(i)$ is the area of the leaves of i whose projection intersects S. Density D can be interpreted as the density of the cuboid above the square and it approximates the percentage of the incoming light that passes through this cuboid. Note that branches that are projected to the same position in the square are both fully considered as we are computing the density of the cuboid.

With this information we need to simulate daylight to compute the illumination within the trees crown. The direction of the incoming sunlight varies over the day. Thus we cannot assume a steady light source with parallel rays, but need to consider directional light. Benes [Ben96] discussed the effect of daylight and its intensity and direction. The light enters the scene with the greatest intensity from the top and decreases towards the sides. Therefore we send many rays from the top and fewer rays from the sides as shown in Figure 3. For interactive modeling, the efficiency of the

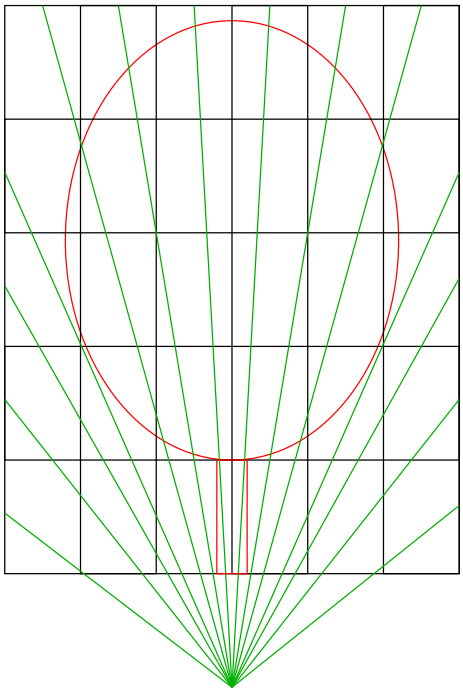


Figure 3: Light rays (green) from the boundary faces directed underneath the tree are used to simulate daylight.

ray traversal computations is important. We adopt the idea of Bresenham's line-drawing algorithm [Bre65] and generalize it to 3D to quickly compute the sequence of cuboids that are traversed by each ray. The amount of light available for each cuboid accumulates with each ray that traverses it. If a ray with intensity I_r hits a cuboid with previous intensity I_c, I_c gets updated using the accumulation rule

$$I_c = 1 - ((1 - I_c) \cdot (1 - I_r)) = I_c + I_r - I_c I_r \ .$$

If the cuboid has density D, then the intensity of the traversing ray I_r is reduced by multiplying it by D. After all ray traversals we have a good approximation of the light available for each cuboid.

4 Rendering

In our implementation, all geometry is rendered as textured polygons. As a texture for the bark, we used photographs of the trees' bark, and as a texture for the leaves, we scanned a typical leaf or took photographs.

 ISBN 978-80-86943-98-5

4.1 Branches

For each branch a closed mesh of generalized cylinders is generated. For branches of higher orders, a new mesh is generated that starts inside the parent branch. Hence, we do not have to compute a special bifurcation structure but instead generate the impression of branches coming out of the parent branch.

4.2 Leaves

The leaves surrounding a branch are rendered as equidistant triangles twisted around the branch. The position of each leaf has to be computed after every growth step as it depends on the new position of the branch and on the new diameter of the branch. For each leaf, the coordinates of a triangle are computed in such a way that the triangles are tangent to the branch. Therefore, no leaf stalk starts inside the branch, and each leaf is connected to the branch. These triangles are rendered using a partially transparent leaf texture.

4.3 Shadows

Shadows are very important for natural-looking scenes. For realistic shadows, we need soft shadowing, which is too time-consuming to compute precisely. Instead, we approximately compute hard shadows and render them using textures with higher opacity in the center and decreasing opacity to the boundary. We can render such polygons in real-time and generate soft shadows.

The positions of the shadows of the branches and the attached leaves are computed using the results of the light computation described in Section 3.4. As a result of the light computation, we obtain planes with projected coordinates of all branch segments. In addition, we store information about their diameters and the leaves at these branch segments. For each projected branch segment, the coordinates of a quadrilateral are computed. For the shadows of the leaves the coordinates of small squares around the respective projected branch are computed. The soft-shadow textures of all branches and leaves are accumulated in a texture buffer, which is rendered at each frame.

Real shadowing includes not only the shadows underneath the tree on the ground but also self-shadowing within the tree. We apply self-shadowing to the leaves with the help of the available light information obtained from the computations described in Section 3.4. The leaves are divided into 10 luminance classes depending on the amount of incoming light. The leaves of each class are rendered with the respective luminance so that leaves appear darker or brighter due to the light that arrives at the small cuboid in which they are located. All leaves are rendered with the same texture.

4.4 Implementation

For every branch segment, the orientation is stored as a rotation relative to the orientation of the previous branch segment. These rotations are not stored as matrices but as quaternions to avoid matrix computations. The branch segments are stored in a binary tree with the first trunk segment being the root. While traversing the tree, the start position of a branch segment is passed as a parameter of the parent branch segment. This position can be used directly, since only the ancestors affect this new position. With this position and the orientation, the coordinates of the generalized cylinders are computed, again with the help of quaternions. Even the leaf positions are computed this way.

The simulation can be stopped at any time for a closer look at the tree of this age. To speed up the performance, we save all the necessary coordinates, normals and texture coordinates in each step and render them at once. Therefore no new geometry computations have to be performed when the simulation is interrupted.

5 Results and Discussion

We animated the growth of prototypical trees from 4 to 50 years. We did not start the animation earlier, as we had no data available for trees younger than 4 years. The animation can be halted at any time to have a closer look at a tree of a certain age. The frame rates we achieved allow for changing the viewpoint interactively even while the animation continues, and growth modeling processes are executed in addition to the rendering.

We were able to animate trees of realistic natural appearance. Moreover, the trees fit the global measured functions of tree height, crown height and width, trunk length, and radius. One can also observe common natural phenomena of trees within our computer-generated models. For example, the interior of the crown exhibits fewer branches and leaves than along the outside due to the light distribution. This shows that using light information for modeling growth really helps improve the model, getting closer to a real tree. The branching structure of the trees looks very realistic. It is not as regular as it would be when applying simple L-systems, and it follows natural laws of tree growth. For the local production rules, we use several stochastic components to achieve a higher diversity within our trees and also among our trees.

In Figure 4, some screen-shots of growing trees taken from the animation can be seen. The trees all have different densities, crown shapes, and overall appearances. The last screen-shot of the two Southern magnolia trees shows the effect randomness has on the trees. Both trees use the same light and growth parameters, but have a distinct appearance. In addition, we provide a video showing the animated growth at `www.math-inf.uni-greifswald.de/informatikJP/rudnick/video`.

Depending on the computer system used, the interac-

 ISBN 978-80-86943-98-5

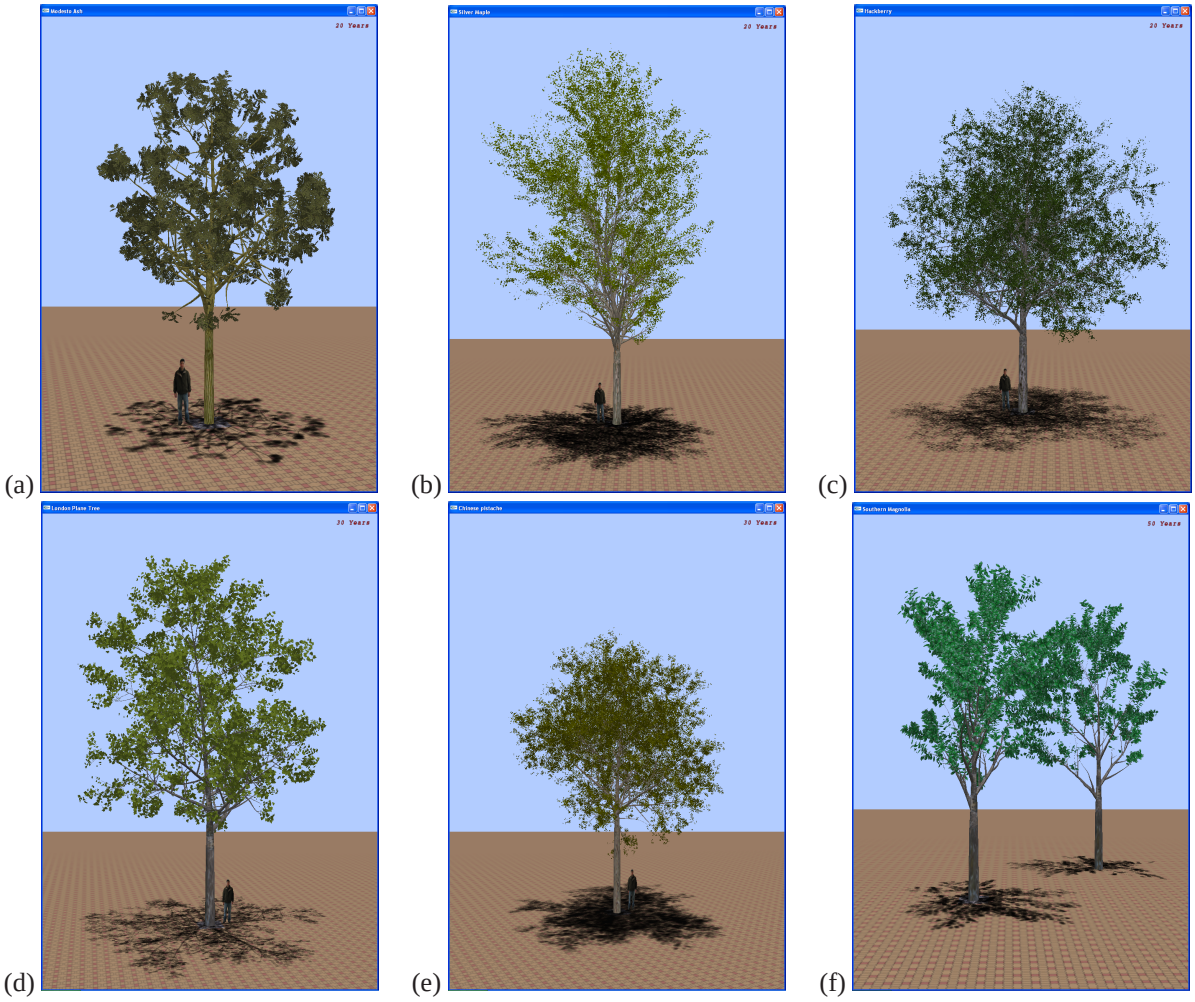


Figure 4: Screen-shots of different tree species taken from our animations: Modesto ash at 20 years (a), silver maple at 20 years (b), hackberry at 20 years (c), London planetree at 30 years (d), Chinese pistache at 30 years (e), and two Southern magnolias at 50 years (f).

tive animation may slow down a little for older trees due to the large number of branch segments. Obviously the time for all calculations depends on the number of branch segments. In every growth step, the local production rules have to be applied to all segments, the light information for every position inside the crown has to be computed, and every branch has to store this information for its own position. For rendering all polygons of branches, leaves, and shadows, the respective geometry has to be computed. When the growth simulation is halted, the animation achieves a better frame rate because no growth steps have to be performed and only the rendering computations are needed. In Table 1, the computation time in milliseconds for the individual modeling computations and the rendering is shown dependent on the number of branch segments. We enlist the computation time for the growth modeling (applying production rules), for updating the light distribution, and for generating the mesh geometry. The Southern magnolia, for example, with 30,000 branch segments is rendered with nearly half a million polygons. The times were measured on a PC equipped with a 3.06GHz processor and an NVidia GeForce 6800 ultra graphics card.

Although implementation on the GPU may be used to further improve the frame rates, our overall goal was to distribute our program over the web for city planning purpose. Thus, our system should run on any personal computer with any hardware configuration. Thus, we decided not to require specific graphics hardware.

6 Conclusions and Future Work

We have presented a real-time tree-growth animation for urban street trees. The real-time computations include all tree-growth modeling steps, geometry generation, and rendering. The growth of the trees is controlled by given, global functions of time for tree height, diameter at breast height, and crown height and width. In our model, we have mapped these global parameters to local production rules. The trees are generated by simultaneously and iteratively applying lo-

 ISBN 978-80-86943-98-5

# branch segments	computation times (in *ms*)			rendering time (in *ms*)
	growth	light	mesh	
5,000	7	7	2	16
10,000	13	8	12	16
20,000	27	12	30	33
30,000	40	13	60	33

Table 1: The computation time in milliseconds for each frame, dependent on the number of branch segments. A tree with 30,000 branch segments is quite a large and old tree. It consists of nearly half a million polygons.

cal production rules to all branches. This approach produces trees with a natural branching structure that fits the global properties. In our model, we have also included the influence of light on the growth of the tree. We used a data structure giving us global control of the tree and allowing us to compute the available light at every position inside the crown at any time. As a result, crown density conforms to characteristics of measured trees, as does crown size.

For rendering purposes, we used a polygon-based approach with photographs and scans for textures. The appearance of the tree is highly realistic when considering real-time constraints. In our animation we reached interactive rates due to the sophisticated yet simple modeling and rendering methods. For shadow computations including self shadowing we made use of the light distributions computed during modeling.

We applied our model to seven different single-stemmed tree species. For future work we plan to apply our method to additional tree species, including multi-stemmed ones. We will also model conifers when the necessary global functions are available. Another objective for future work is to include the effects of buildings on light distribution and tree-growth.

Acknowledgments

This research was supported in part by funds provided by the U.S.D.A. Forest Service, Pacific Southwest Research Station, and the Elvenia Slosson Research Endowment for Ornamental Horticulture at the University of California, Davis.

References

[AK85] M. Aono and T.L. Kunii. Botanical tree image generation. *IEEE Computer Graphics & Applications*, 4(5):10–34, 1985.

[Ben96] B. Benes. An efficient estimation of light in simulation of plant development. In *Computer Animation and Simulation '96*, pages 153–165. Springer-Verlag, 1996.

[Blo85] J. Bloomenthal. Modeling the mighty maple. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 305–311, New York, NY, USA, 1985. ACM Press.

[Bre65] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM System Journal*, 4(1):25–30, July 1965.

[CSHD03] M. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for texture and image generation. In A.P. Rockwood, editor, *SIGGRAPH '03 Proceedings*. ACM SIGGRAPH, 2003.

[DCSD02] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In M. Gross, K. I. Joy, and R. J. Moorhead, editors, *Proceedings of IEEE Visualization '02 conference*. IEEE Computer Society Press, 2002.

[GMW04] C. Galbraith, L. Mündermann, and B. Wyvill. Implicit visualization and inverse modeling of growing trees. *Computer Graphics Forum (Proceedings of Eurographics 2004)*, 23(3), 2004.

[HB03] W. Van Haevre and P. Bekaert. A simple but effective algorithm to model the competition of vitual plants for light and space. *Journal of WSCG*, 11(1), 2003.

[HdFBR04] W. Van Haevre, F. di Fiore, P. Bekaert, and F. Van Reeth. A ray density estimation approach to take into account environmental illumination in plant growth simulation. In *SCCG '04: Proceedings of the 20th spring conference on Computer graphics*, pages 121–131. ACM Press, 2004.

[KK86] R. E. Kendrick and G. H. M. Kronenberg. *Photomorphogenesis in Plants*. Kluwer Academic Publishers, 1986.

[LD99] B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Computer Graphics & Applications*, 19(1), 1999.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development. *Journal of theoretical biology*, 18:280–315, 1968.

[LKMH05] L. Linsen, B. J. Karis, E. G. McPherson, and B. Hamann. Tree growth visualization. *Journal of WSCG*, 13((1)–(3)):81–88, 2005.

[Mat91] C. Mattheck. *Trees: The mechanical Design*. Springer-Verlag, Berlin Heidelberg, 1991.

[Nul68] W. Nultsch. *Allgemeine Botanik*. Georg Thieme Verlag, Stuttgart, 1968.

[PHHM97] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Mech. Visual models of plant development. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. III: Beyond Words*, pages 535–597. Springer-Verlag, Berlin, 1997.

[PHM93] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. In J.T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 351–360. ACM SIGGRAPH, 1993.

[PHMH95] P. Prusinkiewicz, M. Hammel, R. Mech, and J. Hanan. The artificial life of plants: Artificial life for graphics, animation, and virtual reality. In *SIGGRAPH '95 Course Notes*, pages 1–38. ACM SIGGRAPH, 1995.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990.

[PMKL01] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 289–300, New York, NY, USA, 2001. ACM Press.

[SS02] E. Strasburger and P. Sitte. *Lehrbuch der Botanik für Hochschulen*. Spektrum, Akad. Verlag, Heidelberg, 2002.

[SSBD03] C. Soler, F.X. Sillion, F. Blaise, and P. Dereffye. An efficient instantiation algorithm for simulationg radiant energy transfer in plnat models. *ACM Trans. Graph.*, 22(2):204–233, 2003.

Fast Skeleton Estimation from Motion Capture Data using Generalized Delogne-Kåsa method

Jonathan Kipling Knight and SK Semwal
Department of Computer Science
University of Colorado
Colorado Springs, CO. 80933, USA
kip.knight@itt.com | semwal@cs.uccs.edu

ABSTRACT

This paper presents a fast closed-form solution estimating the rotation points of joints relative to the motion capture data. The proposed solution estimates the physical location of joints inside the body of the person wearing the trackers. The Generalized Delogne-Kåsa method used for our implementation fits spheres, cylinders, circles and planes to the motion capture data without the need for initial guessing. This non-iterative, closed-form solution is fast as it calculates the rotation point with O(N) averaging along with one inversion of a 3x3 positive semi-definite matrix for each joint. The error in the joint location is on average $3\sigma/\sqrt{N}$ which is low. In addition, sample points for every joint can be from different time sequence allowing flexibility in recovering the joint locations. Once the joint location relative to the tracker position is determined, it could be used for the remainder of the data set. Publicly available CMU motion capture data was used for this study. Two animation sequences, showing our method, are included with this paper. These results can be compared to that available at the CMU site for the same animation. Since the pose is found relative to the given data, our pose estimation provide better fit to the given data, revealing subtle, individual nuances of the person used for the motion capture. Because of the closed form solution, our technique is ideally suited for the use of motion captured data to create skeletal motion in 3D games or applications where real time performance is essential.

Key words: sphere curve fit, joint location, articulated motion, Delogne-Kåsa Method, motion-capture data

1. Introduction

Motion capture animation has been continuously improved by many authors [2, 5, 6, 12-21, 25, 27, 28, 32]. Their studies can be divided into two methods of analysis: kinematic and kinetic [1]. In kinematic methods, scientists study the mechanical displacements of the limbs during motion. In kinetic (or dynamics) methods, the energies and forces on the limbs are studied during the motion of the articulated figure.

Kinematic methods are used in animation by determining the joint angles from space-time constraints. Holt et al. [8] estimated the 3D motion of an articulated object from a sequence of 2D perspective views. They used a decomposition approach to break down the motion of each segment. This was a good use of video motion capture to estimate the animation of a figure. Choi [5] improved standard forward kinematic techniques to provide more accurate end-effector positions. Inverse kinematics is another technique employed by many authors [19]. Kinematics tends to be a faster technique than kinetics.

Kinetic methods consider the changes in energy, inertia, and forces to create animation. These changes determine the way the joint angles change in time. Semwal et al. [24] visualize the leg rotations of a cyclist showing forces during the pedal-movement. A straightforward approach is to use Newtonian or Lagrangian mechanics, and involves solving simultaneous second order partial differential equations. Recursive methods

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

 ISBN 978-80-86943-98-5

can be more efficient [30]. Liu and Popović [15] infuse physical reality in to sparsely keyed motion data. An articulated figure realistically played hopscotch using a minimal set of pre-determined positions. Liu and Popović [15] relied on off-line calculations. Popović and Witkin [21] discuss a novel application by transforming standard models of motion in to a diverse assortment of similar motions. For instance, a standard run sequence can transform into a run with a limp-sequence, while retaining realism.

Recently, motion captured data has become a rich source for providing realistic human motion. Content based retrieval [17] and using lower dimensional data [5] are recent examples of use of the motion captured data in recent studies. Kinetics is of interest on the motion captured data in Zordan [32]. Most of the existing methods involve an initial guess of the underlying skeleton and iterating through until a stable solution evolves. Various forms of least squares fitting are the most popular, most of which use the Levenberg-Marquart method for solving the minimum. O'Brien [19] calculates a skeleton and then uses inverse kinematics to produce motion from motion capture data. Our method also belongs to the kinematics group by analyzing motion-captured data to find a skeleton within the data similar to O'Brien [19]. The proposed method improves on the work of O'Brien [19] by providing a faster replacement for the least squares fit of joints. Our technique is more computationally efficient than O'Brien because once the offsets from the sensors are calculated during pre-processing, they are reused during skeletal animation. This direct approach captures the subtle deformations that the human body is capable of undergoing during complex movements and is inherent in the sensor (motion captured) data. These subtle body deformations are also present in our skeletal animation because of our algorithm, thus improving the quality of animation. Although not a focus of this paper, an incremental improvement technique, being implemented, would provide yet another order of magnitude improvement in future.

An articulated figure can be divided into rigid *segments* that are connected to each other by joints. A tree structure is a perfect way to organize these segments [19]. Root to leaf processing of segment properties allow for a dependency on the parent segment's position and orientation. Such a dependency is exploited in this paper. The tree structure of segments can be known ahead of time (a-priori). When the tree structure is not known, e.g. motion data from an unknown figure), much more off-line analysis is needed to produce the hierarchy. The solution is the equivalent of a minimal spanning tree [19]. Ko and Cremer [11] used 34 DOF for their a-priori system. Bolt [4] used 17 DOF in his model of the human figure. The proposed research in this paper extends to arbitrary number of DOF, and motion captured from other animal forms. Although implemented for a human skeleton that conforms to a tree hierarchy, our technique easily extends to the case when tree structure is not apparent, and animation must directly work from the sensor data.

2. Closed Form Rotation Point Determination

In our research, it was found that a closed-form solution existed when the variance of the square of the distance from the measurement to the point of rotation is minimized. Zelniker [31] shows that the 2D equivalent had been studied and rediscovered since the 1970's. Zelniker generalized the method to any number of dimensions and proved that the bias in the estimation is not significant enough. Our Monte Carlo experiments comparing the Least Squares Linear Regression (LS), and the Generalized Delogne-Kåsa Estimator (GDKE) show them to have approximately the same error in answer. The speed is improved when analyzing the same count as shown in Figure 1.

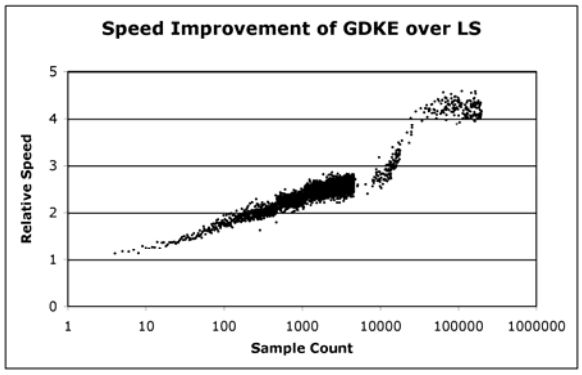


Figure 1: Relative speed improvement of GDKE versus linear least-squares.

Previous solutions [3, 7] and recently O'Brien et.al. [19] determine the rotation point by iterating on a least-squares equation or M-estimators starting from an initial guess. This approach involves either linear or non-linear fitting of the data and has a chance of not converging to a solution. The initial guess must be close enough to the truth or the iterations may diverge away from the point. The GDKE solution to the center of rotation involves no guessing, no iterations, and the inversion of a 3x3 matrix. The GDKE is robust with noise and also can distinguish cylindrical joint motion, with a little extra work. The speedup shows that our method is computationally more efficient than the least-

 ISBN 978-80-86943-98-5

square method of O'Brien's [19], and is a suitable replacement for both linear and non-linear least-squares fitting of a sphere, cylinder, circle, and a plane.

The requirements in order to determine the point of rotation for this method are: (a) Known fixed axes on a parent segment. (b) No translational movement. (c) Enough rotational motion. (d) At least 4 positions of one marker. The above requirements are satisfied for the problem in hand which amounts to finding the best-fit sphere for a 2 or 3 degrees-of-freedom (DOF) joint or the best-fit cylinder for a 1 DOF joint. According to the National Institute of Standards and Technology [26] the best approach to this problem is non-linear Least-Squares fitting but that method could be problematic for the general case. O'Brien [19] uses linear least-squares fitting. Our research, consistent with recent results in Zelniker's [31], shows that the GDKE is a viable and robust replacement for speeding up the calculations as explained below.

2.1. Theory

Presented here is a derivation of the Generalized Delogne-Kåsa Estimator (GDKE). The estimator is a closed form solution for a hypersphere that has O(N) averaging and involves the inversion of a 3x3 positive-semidefinite matrix. Stated simply -- the solution involves solving for the absolute minimum of the variance of the square of the lengths from the point of rotation to each position for a single marker. The ith position for the jth joint (x_{ji}) around the joint's rotation point (c_j) can be expressed as

$$x_{ji} = c_j + R_{ji}\rho_{ji}$$

where

$$\left|\rho_{ji}\right| = 1$$

This position (x_{ji}) is not the absolute position that comes directly from motion capture. Instead it is relative to the parent segment in its fixed coordinate system. The square of the radius of the measurement (R_{ji}) has a variance of

$$s_j^2 = \tfrac{1}{N_j - 1}\sum_{i=1}^{N_j}\left(R_{ji}^2 - \hat{R}_j^2\right)^2 \qquad (1)$$

where the radius can be written as

$$R_{ji} \equiv \left|x_{ji} - c_j\right|$$

and the average square of the radius is

$$\hat{R}_j^2 = \tfrac{1}{N_j}\sum_{i=1}^{N_j} R_{ji}^2 .$$

Equation 1 is then minimized to solve for the GDKE rotation point (c_j). Setting the gradient of the variance to zero thus

$$\nabla s_j^2 = 0$$

will provide an equation to find c_j. The gradient can be written, after much algebraic manipulation not presented here due to space-limitation, as

$$\nabla s_j^2 = 4\left(2\mathbf{C}_j\left(c_j - \bar{x}_j\right) - \mathbf{S}_j\right)$$

where $\mathbf{C}_j$ is the standard definition of the sample variance-covariance matrix for a vector quantity given by

$$\mathbf{C}_j = \tfrac{1}{N_j - 1}\sum_{i=1}^{N_j}\left(x_{ji} - \bar{x}_j\right)\left(x_{ji} - \bar{x}_j\right)^T \qquad (2)$$

The average is given by

$$\bar{x}_j = \tfrac{1}{N_j}\sum_{i=1}^{N_j} x_{ji} . \qquad (3)$$

The variance-covariance matrix (2) is a biased estimator for the covariance of the vector measurement. $\mathbf{C}_j$ has been used in many situations [9] for curve fitting and has proven useful in identifying the best-fit plane for data.

The other vector quantity $\mathbf{S}_j$ can be considered the vector equivalent of the third central moment and is

$$\mathbf{S}_j = \tfrac{1}{N_j - 1}\sum_{i=1}^{N_j}\left(x_{ji} - \bar{x}_j\right)\left(x_{ji} - \bar{x}_j\right)^T\left(x_{ji} - \bar{x}_j\right) \qquad (4)$$

Equation 4 is an unbiased mix of multi-dimensional moments, and is a new quantity, not available in [29, 23]. The final solution for the center of the hypersphere (i.e. the relative joint location) is

$$c_j = \bar{x}_j + \tfrac{1}{2}\mathbf{C}_j^{-1}\mathbf{S}_j . \qquad (5)$$

This formula looks different than Zelniker [31] but is algebraically equivalent.

The variance-covariance matrix $\mathbf{C}_j$ is positive-semidefinite so Cholesky decomposition provides a more efficient solution to the equation. There are two cases that Cholesky decomposition produces an undesirable answer or even fails. The trivial case is if all points coincide, then $\mathbf{C}_j$ is singular. This case occurs

 ISBN 978-80-86943-98-5

when the joint does not move. The non-trivial case is during planar motion. If there exists a vector n such that

$$x_{ji}^T n = k$$

where k and n are constants for all positions of the marker, then $\mathbf{C}_j$ is singular. All is not lost though with these singularities or even near singularities. The planar motion can still be solved for as explained below.

The Null Space of a matrix is a set of vectors that solve the equation $\mathbf{C}_j n=0$. This set of vectors is inherently extracted during the Singular Value Decomposition (SVD) [22] of a matrix based on some threshold. The condition number of $\mathbf{C}_j$ will provide the threshold and the measure for which to check if the motion is planar. The condition number is the ratio of the largest eigenvalue to the smallest eigenvalue. If the threshold is equal to the inverse of the condition number then a single vector exists in the Null Space of $\mathbf{C}_j$ and that vector happens to be the normal to the plane of motion. That vector is also the eigenvector that corresponds to the smallest eigenvalue. This is proven by solving for the minimum of the variance of the distance from the plane. The distance (z_{ji}) from the plane for each point is

$$z_{ji} = x_{ji}^T n.$$

The gradient of the variance is then determined to be

$$\nabla Var\left(z_j\right) = 2\mathbf{C}_j n$$

which clearly shows the minimum occurs when $\mathbf{C}_j n=0$. The only solution to this equation is the Null Space of the variance-covariance matrix $\mathbf{C}_j$. If the motion is determined to be planar the null vector can be used to determine the center of the circle on the plane. The equation for the best-fit circle is then

$$c'_j = c_j + nn^T\left(\bar{x}_j - c_j\right) \tag{6}$$

which basically removes one dimension along the null vector for the solution to the hypersphere.

So we have a solution for the equation when it is both singular and positive. How do we know it is unique or worse yet, maybe it is a maximum and not a minimum? The answer can be achieved by finding the double derivative of s^2_j (i.e. the Hessian). The Hessian is determined as

$$\tfrac{1}{2}\nabla\nabla^T s_j^2 = 4\mathbf{C}_j.$$

Since the Hessian is positive-semidefinite, the answer in Equation 1 is proven to be an absolute minimum.

2.2 Monte Carlo Simulations

The GDKE method estimator has been shown to be biased [31]. Zelniker's analysis showed that the bias is of order of the standard deviation of the measurements. This bias is actually not significant enough to warrant dismissal of the method. We conducted Monte Carlo simulations of spherical data to confirm our claim. Data points were produced uniformly on a sphere with a known error introduced to the positions. The number of points were uniformly chosen between 4 and 1000 inclusive. The known error was picked with a Log-Normal(0,10) distribution. The radius of the sphere was chosen with a LogNormal(0.18,1). The center of the sphere was chosen with Gaussian(0,0.5) uniformly around the point (0.6,-0.2,0.9). The simulation was run 5000 times and GDK estimators were calculated. The bias from the true center was analyzed using the Ordered Statistics method [10] of determining the probability distribution. The data showed that the bias magnitude is proportional to $\sigma/\sqrt{N}$. The ordered statistics method shows a best fit probability of Weibull(2.378, 3.302) as can be seen in Figure 2.

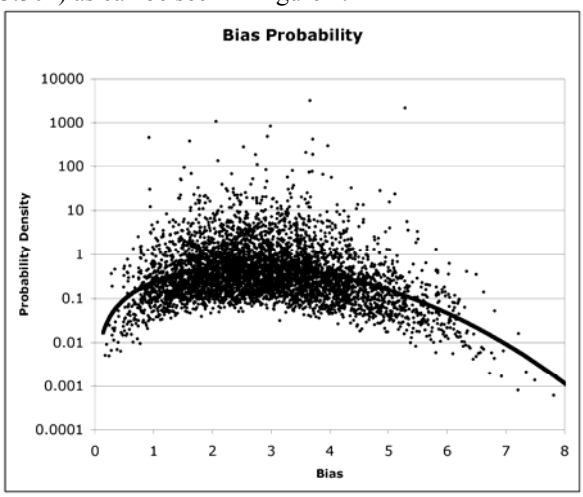


Figure 2: Probability Density for the bias for the GDKE center.

The resultant distribution has an expected value of 2.93 and a standard deviation of 1.31. So the final center estimator for the sphere can be expressed empirically as

$$\left|c - c_0\right| \approx 2.93\frac{\sigma}{\sqrt{N}} \tag{7}$$

which is $O(\sigma/\sqrt{N})$ where N is typically 100 samples during preprocessing. Here, c_0 is the true center, σ is the true standard deviation of the positions, N is the

 ISBN 978-80-86943-98-5

sample count. The radius was similarly analyzed to find that the value is unbiased when

$$\sigma < 0.1R_0$$

as shown in Figure 3. R_0 is the true radius. This situation occurs for most measurements and makes physical sense since one does not want to have an inaccuracy anywhere near the radius of the sphere (i.e. $\sigma = R_0$). The empirical formula for this situation is

$$\left|R - R_0\right| \approx 0.$$

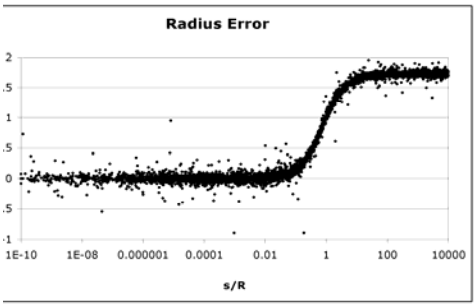


Figure 3: Error in GDKE radius estimate

These experiments and the analysis from Zelniker show that the GDKE method is a better replacement for the slower linear and non-linear least-squares fitting used by most authors, including OBrien et al. [19].

2.3. Procedure for Determining Center

The procedure to determine point of rotation of a joint is as follows. This method automatically determines the type of joint, either spherical, cylindrical or not moving at all.

1) Choose one marker with absolute positions x'_{ji} on segment.
2) Make points relative to parent.
$\mathbf{M}_{j-1}$ = column matrix of parent axes
p_{j-1} = center of parent's coordinate frame

$$x_{ji} = \mathbf{M}_{j-1}^{T}\left(x'_{ji} - p_{j-1}\right)$$

3) Calculate mean, covariance and third central moment of points using Equations 2,3,4.
4) Using singular-value decomposition, determine the condition number, the null-space vector, and the solution for the best-fit sphere. Use Equation 5.
5) If condition number is small then use spherical solution otherwise find the rotation center for the planar motion. Use Equation 6.
6) Store the center of rotation and the null-space vector for later use of displaying skeleton at each frame.

2.4. Applying the GDKE to Motion captured Data

The motion capture data used in this paper comes mainly from the very large database (>2GB) of motions captured by Carnegie Mellon University Graphics Lab (CMU Labs). The data is freely downloadable at http://mocap.cs.cmu.edu/. The database was created with funding from the NSF grant EIA-0196217. There are 1576 trials in 6 categories and 23 subcategories. We also analyzed the data from Advanced Computing Center for the Arts and Design (ACCAD) at http://accad.osu.edu/.

Motion capture information is acquired on up to forty or more markers on the body depending on the motion being studied. The CMU data comes in a few file formats. The raw Cartesian coordinates of the data are stored in the C3D file format [18]. Each time frame is stored, with each frame consisting of X,Y,Z for each marker on the body. If any data is missing from a frame, that point is zeroed and marked. The captured motion also comes in the form of ASF and AMC file formats. These are created after the VICON program has done analysis of the data. The ASF format stores the skeleton and joint information to create an articulated figure on the screen. The AMC format contains every time frame's translation and rotation for each bone. Our method works with the raw XYZ data and therefore does not consider the post-analysis data in the AMC files. A motion capture file (e.g. C3D file) contains the data, as well as a name associated with each marker. For example, a marker is placed on the right thigh and is called "JOE::RTHI". Usually, the animator doesn't have the same designation so a cross-correlation must be achieved to identify which segment this marker belongs. Our implementation uses a two-file process to cross-correlate a C3D data-set with the segments on the ATM model.

2.3. Extracting Joint Information in Our Implementation

Most information in motion capture data comes in the form of Cartesian coordinates of markers placed on the segments of an articulated figure. Each joint between segments has a point of rotation (if purely rotational) and joint axes. Magnetic tracking sometimes comes with orientation axes so much of this section can be skipped if that is the case. Usually, and in our case, the frames of data must be analyzed during joint motion to determine the axes and rotation point. The center of rotation can be calculated for each joint if: 1) The orientation of the parent segment can be evaluated. 2) The

 ISBN 978-80-86943-98-5

child segment has at least one marker. 3) The child segment is in motion relative to the parent segment.

The orientation of a segment can be determined if there are at least three non-linear data points fixed to that segment. One of the data points can be the center of rotation. There is a hierarchical dependency for determining the center of rotation and the orientation. Thus, the orientation can be determined if there: 1) is one data point, rotation point, and a rotation axis; 2) are two data points and one rotation point, all non-collinear; 3) are three non-collinear data points. The information can be retrieved if calculated hierarchically from root to leaf. First, define the tree. Then, assign the markers to their appropriate segments. The root must have at least three markers. No center of rotation for the root can be determined. The children of the root can determine their center of rotation relative to their parent by the above mentioned method of GDKE.

The rotation point c_{jk} for the current time frame k is determined from

$$c_{jk} = x_{jk} + \mathbf{M}_{jk} v_j$$

where v_j is the local relative vector as determined from GDKE, x_{jk} is the center of the coordinate system, usually one of the marker points, and $\mathbf{M}_{jk}$ is the 3x3 matrix of column vectors that represent the axes of the segment's coordinate system.
The coordinate matrix is composed of three column vectors that make the orthogonal coordinate system. It is determined differently for each of the three cases.
1) One data point is available, use null-vector:

$$\hat{x} = \hat{n}$$

2) Two data points are available, use second point and center of rotation:

$$\hat{x} = \frac{p_1 \times c}{|p_1 \times c|}$$

3) Three data points are available, use second and third point:

$$\hat{x} = \frac{p_2 \times p_1}{|p_2 \times p_1|}$$

Then the other two axes can be calculated from the first point by

$$\hat{z} = \frac{p_0 \times \hat{x}}{|p_0 \times \hat{x}|}$$

and

$$\hat{y} = \hat{z} \times \hat{x}$$

These produce the rotation matrix **M** by placing them in the columns.

2.6. Generated Motion

The root segment (hips) motion can be considered freely moving and freely rotating. In order to find the motion of such an object (6 DOF), the segment needs at least three unique points on the surface tracked. The motion capture data can be used to create a set of orientation axes and produce the center for any particular frame of the data. The axes are fixed relative to the root segment and are a 3x3 matrix determined by the above equations. Next we use the motion data set and recursively traverse the tree to traverse down the segments. Each segment in the tree contains its own time-dependent position relative to its parent. To position the figure at a specific time-frame involves traversing the tree from the root segment to the endpoints. During the analysis phase, data will be collected giving each segment a chance to calculate its motion constants. Once these constants of the data-set are determined (i.e. relative rotation points and null-vectors), skeletons for the remainder of a motion capture data are easily drawn using these determined constants. This is computationally far efficient than applying the inverse-kinematics solution in existing techniques.

3. Results

We conducted several experiments comparing GDKE method with linear least square method and found that the GDKE produces similar answers to the Least-Squares Method of O'Brien et al. [19] but is computationally more efficient as discussed in the paper. In addition, this new method can also produce a better answer in the cylindrical case. The error in the calculation is only of $O(\sigma/\sqrt{N})$ [31] and corresponds to the bias as explained in the empirical relationship in Equation 7.

An example of a single time-frame for a karate pose is given in Figure 4. A more complex data-set was analyzed involving two figures during a salsa dance. A time-frame from that data is given in Figure 5.

 ISBN 978-80-86943-98-5

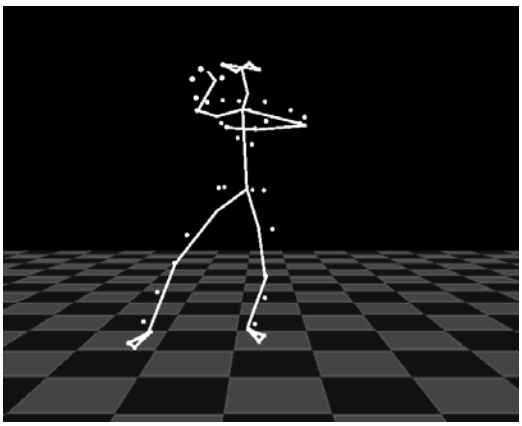


Figure 4: Karate Pose from ACCAD data "Break Dancer 1"

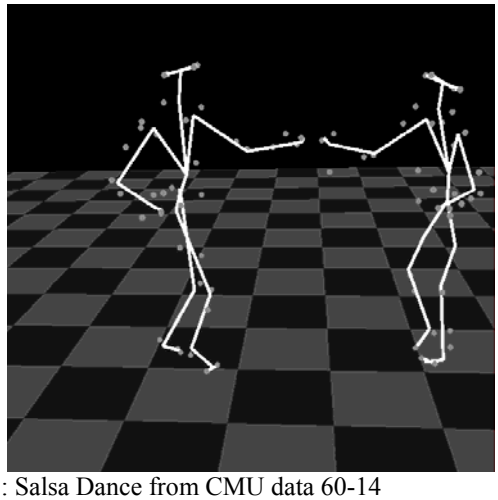


Figure 5: Salsa Dance from CMU data 60-14

Noise can also contribute to the propagation of errors. Noise comes in two flavors, measurement noise and process errors. A look at the problem at hand, i.e. measured markers on a body, shows how errors are introduced. The accuracy of capturing the coordinates of the marker in time is considered measurement noise. This is usually a simple Gaussian noise for each measurement. Process errors are produced when, for example, the marker is placed on loose clothing and subsequently moves slightly on the body during motion. All these errors contribute to the accuracy of the rotation point calculation whichever method is chosen.

4. Conclusions and Future Research

We have presented a GDKE implementation that is computationally superior to the method of O'Brien et al. [19]. In addition, our method is faster that inverse kinematic solutions as the actual joint locations are directly calculated from the sensor data. These estimates are found by using a fast closed form solution. As our method derives the animation from sensor data, animation sequences preserve the subtle variations of the person used for the motion capture study. Qualitatively, this can be observed while playing the CMU data and our animation side by side. Our algorithm is well suited for on-the-fly capture and display of the motion capture data, and would allow merging of animation sequences with ease. In the future, we plan to utilize an incremental improvement technique, which could further provide an order of magnitude speed improvement.

Acknowledgements

CMU motion capture data sets were essential for our study. We gratefully acknowledge all who were involved in collecting this invaluable resource that was publicly available to us. Our "break-dance" and "salsa" movies correspond to the CMU data-set 85-14 and 60-14 respectively. Other data-sets analyzed were from ACCAD.

References:

[1] J.K. Aggarwal, Q. Cai, and W. Liao, "Nonrigid motion analysis: articulated and elastic motioin" *Computer Vision and Image Understanding* 70, no 2 (May 1998) : 142-56.

[2] J. Assa, Y. Caspi, D. Cohen-Or, "Action Synopsis : Pose selection and illustration" *ACM Transaction on Graphics (SigGraph 2005)* 24(3) (July 2005) : 667-676.

[3] B. Bodenheimer, C. Rose, S. Rosenthal, J. Pella, "The Process of Motion Capture: Dealing with the Data" *Eurographics* CAS (Sept. 1997): 3-18

[4] D.A. Bolt, "Two Stage Control for High Degree of Freedom Articulated Figures" Ph.D. thesis for Computer Science, University of Colorado in Colorado Springs, (2000).

[5] J. Chai, J. Hodgins "Performance Animation from Low Dimensional Control-Signals" *ACM Transaction on Graphics (SigGraph 2005)* 24(3) (July 1005) : 686-696.

[6] K. Choi, S. Park, H. Ko. Processing motion capture data to achieve positional accuracy. *Graphical Models and Image Processing*, 61(5):260–273, September 1999.

[7] L. Herda, P. Fua, R. Plankers, R Boulic, D. Thalmann, "Skeleton-based motion capture for robust reconstruction of Human Motion", Computer Graphics Lab (LIG) EPFL:Lausane, Switzerland (2000).

[8] R.J. Holt, T.S. Huang, and A.N. Netravali, R.J. Qian, "Determining articulated motion from perspective views: a decomposition approach" *Pattern Recognition*: 30, no. 9,

(1997): 1435.

[9] K. Kanatani. "Cramer-rao lower bounds for curve fitting." *Graphical Models and Image Processing*, 60(2):93–99, March 1998.

[10] J. K. Knight. Probability Density Function Determination from Ordered Statistics. Masters Thesis for Applied Mathematics, California State University, Fullerton, August 1995.

[11] H. Ko, J. Cremer, "VRLOCO: Real-Time Human Locomotion from Positional Input Stream" *Presence* 5(4) (1996): 367-380.

[12] L. Kovar, M. Gleicher, F. Pighin, "Motion Graphs" *Proceedings of SIGGRAPH 2002*: 473-482.

[13] J. Lee, S.Y. Shin, "A Hierarchical Approach to Interactive Motion Editing for Human-like Figures" *Proceedings of SIGGRAPH 1999.*

[14] J. Lee, J. Chai, P.S.A Reitsma, "Interactive Control of Avatars Animated with Human Motion Data" *Proceedings of SIGGRAPH 2002*: 491-500.

[15] C.K. Liu, Z. Popović, "Synpaper of Complex Dynamic Character Motion from Simple Animation" *Proceedings of SIGGRAPH 2002*: 408-416.

[16] Z. Lui, S. Gortler, M. Cohen, "Hierarchical Spacetime Control" Computer Graphics (*SIGGRAPH 1994 Proceedings*).

[17] M. Muller, T. Roder, M. Clausen, "Efficient Content-Based Retrieval of motion capture Data" *ACM Transaction on Graphics (SigGraph 2005)* 24(3) (July 2005) : 677-685.

[18] Motion Lab Systems, Inc. "C3D Format User Guide", PDF document at http://www.motion-labs.com, Motion Lab Systems, Baton Rouge, LA, (2003) (98 pages)

[19] J. F. O'Brien, R. E. Bodenheimer Jr., G. J. Brostow, and J. K. Hodgins. *Automatic Joint Parameter Estimation from Magnetic Motion Capture Data*, pages 53–60, Montreal, Quebec, Canada, May 15-17 2000. Graphics Interface.

[20] N.S. Pollard, P.S.A. Reitsma, "Animation of Humanlike Characters: Dynamic Motion Filtering with a Physically Plausible Contact Model" *Yale Workshop on Adaptive and Learning Systems* (2001).

[21] Z. Popović, A. Witkin, "Physically Based Motion Transformation" *Computer Graphics Proceedings*, Annual Conference Series, 1999.

[22] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, "Numerical Recipes in C: The Art of Scientific Computing" 2nd.ed., Cambridge University Press: 1992.

[23] P. L. Rosin. Further five point ellipse fitting. *Graphics Models and Image Processing*, 61:245–259, 1999.

[24] S. Semwal, M.J. Parker, "An Animation System for Biomechanical Analysis of Leg Motion and Predicting Injuries during Cycling" *Real-Time Imaging*: 5 (1999): 109-123.

[25] S.K. Semwal, R. Hightower, S. Stansfield, "Closed Form and Geometric Algorithms for Real-Time Control of an Avatar" *IEEE Proceedings of VRAIS 1996*.

[26] C.M. Shakarji "Least-Squares Fitting Algorithms of the NIST Algorithm Testing System" *J. of Research of the National Institute of Standards and Technology*: 103, No. 6 (1998): 633.

[27] D. Tolani, A. Goswami, N.I. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs" *Graphical Models*: 62 (2000): 353-388.

[28] M. van de Panne, "From Footprints to Animation" *Computer Graphics Forum*: 16, no. 4 (1997): 211-223.

[29] L. Yang, F. Albregtsen, and T. Taxt. "Fast computation of three-dimensional geometric moments using a discrete divergence theorem and a generalization to higher dimensions." *Graphical Models and Image Processing*, 59(2):97–108, March 1997.

[30] J. Znamenáček and M. Valášek. An efficient implementation of the recursive approach to flexible multibody dynamics. *Multibody System Dynamics*, 2(3):227–251, 1998.

[31] E. E. Zelniker and I. V. L. Clarkson. A Generalisation of the Delogne-Kåsa Method for Fitting Hyperspheres. Pacific Grove, California, November 2004. Thirty-Eighth Asiomar Conference on Signals, Systems and Computers.

[32] V.B. Zordan, A. Majkowaska, B. Chiu, M. Fast, "Dynamic Response for the Motion Capture Animation" *ACM Transaction on Graphics (SigGraph 2005)* 24(3) (July 2005) : 686-701.

[33] A Kirk, J O Brien, and D Forsythe. "Skeletal parameter estimation from optical motion capture data," Proceedings of IEEE conference on Computer Vision and Pattern Recognification (CVPR), 2005, pp. 782-788.

Tweakable Shadows for Cartoon Animation

Hidehito Nakajima
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo Japan, 169-8555
nakajima@mlab.phys.waseda.ac.jp

Eiji Sugisaki
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo Japan, 169-8555
eijil11@toki.waseda.jp

Shigeo Morishima
Waseda University
3-4-1 Okubo, Shinjuku-ku,
Tokyo Japan, 169-8555
shigeo@waseda.jp

ABSTRACT

The role of shadow is important in cartoon animation. Shadows in hand-drawn animation reflect the expression of the animators' style, rather than mere physical phenomena. However shadows in 3DCG cannot express such an animators' touch. In this paper we propose a novel method for editing the shadow with both advantages of hand drawn animation and 3DCG technology. In particular, we discuss two phases that enable animators to transform and deform the shadow tweakably. The first phase is that a shadow projection matrix is applied to deform the shape of the shadow. The second one is that we manipulate vectors to transform the shadow such as scaling and translation. These vectors are used in shadow volume method. The shadows are edited directably by integration of these two phases. Our approach enables animators to edit the shadow by simple mouse operations. As a result, the animators can not only produce shadows automatically but also reflect their style easily. Once the shape and location of shadow are decided by animators' style in our method, 3DCG techniques can produce consistent shadow in object motion interactively.

Keywords

Cartoon animation, Shadows, Non-photorealistic rendering, Cel animation, interactive techniques

1. INTRODUCTION

Since the natural shape and animation of shadow are tedious to draw by hand, shadowing in traditional cel animation requires a lot of manual labor. However, shadows play an important role in cel animation as well as various other animations. The shadow is not so outstanding as to attract viewer's attention such as facial animations or human motions, but sometimes the expression of shadows gives a specific meaning to a scene. Especially, in hand-drawn cel animations, a shadow is drawn according to animators' style, which may not be physically correct but looks natural and fantastic. For instance, when animators intend audience to focus on a character's face, animators may edit the shadow covering character's face, may draw simple shadows or may not express the shadow. Likewise, animators who want to express impressive scene draw shadows as if the shadow itself were a lively character or a terrible monster.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright UNION Agency – Science Press, Plzen, Czech Republic.

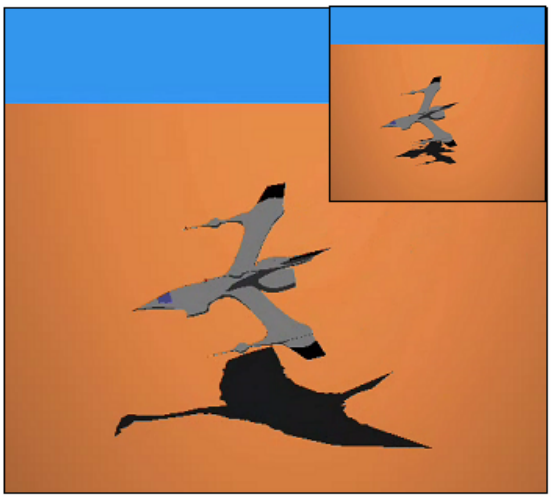


Figure 1. An example of impressive shadow by our tool. The shadow becomes a different shape from the original object. (in this example, the shadow is like a bird) Inset: the original shadow before editing the shadow.

However, it is difficult for the animator to express their styles interactively by using 3DCG technology. Since the shadow in 3DCG is produced automatically by calculating physics based illumination, the procedure for creating 3DCG shadows has a

 ISBN 978-80-86943-98-5

constraint that depends on the relationship between the light source and a 3D model, and this is why shadows in 3DCG have been considered difficult to edit their shapes, scales, and positions. When animators want to change or control shadows in 3DCG, they only can correct the position of light source, modify 3D models indirectly so far. These tedious operations might disturb animators' creativeness.

At the same time, there are many advantages of using 3DCG methods. For instance, we can check an animation result from any kinds of views for animatics. Using a 3D model enables animators to redraw and recreate scene easily, and all processes except setting environmental or modeling parameters are automatically performed. These advantages reduce animators' manual labor. Nowadays, in the cartoon animation industry, the technique of combining 3DCG method and 2D hand-drawn animation has been introduced [Kri06a].

Therefore, we propose the new world that leads to animators to bring out creative mind for editing shadow by combining animators' style and 3DCG technique. Figure 1 shows impressive shadow created by our proposed tool.

Our method is divided into two phases, i.e., deformation and transformation. First, an applied shadow projection matrix is used to deform the shape of the shadow. Second, manipulating vectors of the light obtained by shadow volume method enables transformation of the shadow. By using our method, shadows are considered as the 3D model that animators can transform and deform. These two methods enable animators to hybrid with both animators' style and 3DCG technology. By combining both advantages, animators can edit shadows in 3DCG to apply to their style, and add shadows even into the cel animation scene. Additionally, our tool can apply to key-frame animations. Therefore animators can edit not only shadows in an image but also in key-frame animation. Our method improves working efficiency and helps to express animators' style by using 3DCG technique.

2. RELATED WORK

In this section, we discuss the overall of the computer graphics techniques in cartoon animation. Moreover we mention related work of light and shadow design, and interactive rendering in non-photorealistic field.

[Las87a] was likely the first researcher to discuss the principles of traditional 2D hand-drawn animation. He mentioned these principles applied 3D computer animation. In his paper, he described how to make the audience entertain and presented this method to animators.

Recently, 3D computer graphics techniques have been used in the cartoon animation industry (e.g. Ghost in the Shell). The movie "Appleseed" was a landmark animation featuring hyper realistic imagery and a hybrid 2D and 3D style [Kri06a].

[Cor98a] was a method for applying complex textures on 3D objects to hand-drawn animation. [Rad98a] presented a technique called "View-Dependent Geometry". In conventional cel animation, characters or objects often have all kinds of view-dependent distortions that cannot be described with 3D models. Thus they capture the view-dependent distortions as the reference drawing. [Sug06a] proposed cartoon-like hair motion creating technique. Motion cloning from cartoon hair animation sequences enables animators to create cartoon hair animation easily. [Jue06a] presented the "Cartoon Animation Filter". This filter converts an arbitrary input motion signal to output motion that is more "alive" or "animated". [Chr06a] provided the re-using method for traditional animations. First, they divide cartoon images from their backgrounds. Then, they provide inbetween contours and textures.

As for light and shadow techniques, [Bar97a] described various lighting and shadowing techniques for computer cinematography. They described that the purposes of lighting for cinematography are to contribute to the storytelling, mood, and image composition, and to direct the viewer's eye. These factors are important for cartoon animation same as cinematography. [Pet00a] proposed to inflate a 3D figure based on 2D hand-drawn images to produce shadows semi-automatically for cel animation. Our method focuses on flexibility compared to their method. [Pel02a] proposed interactive interface for transforming the shadow directly on shadowed region. However, this method did not solve the constraint that shadows linked to relationship between lights and objects. In our paper, we try to solve this problem. [Lak00a] presented several real-time methods to emulate cartoon styles, but they did not describe the shadow technique. [Anj03a][Anj06a] proposed intuitive highlighting technique. This method enables animators to edit stylized highlight directly on a 3D object surface. This concept is applied to shadows in our method.

3. SHADOW EDITING PROCESS

In this paper, we introduce two phases for editing shadows. In the first phase, we create an object that represents the shape of a shadow by using shadow projection matrix. We call this object shadow object. We set the shadow object precisely in the same

 ISBN 978-80-86943-98-5

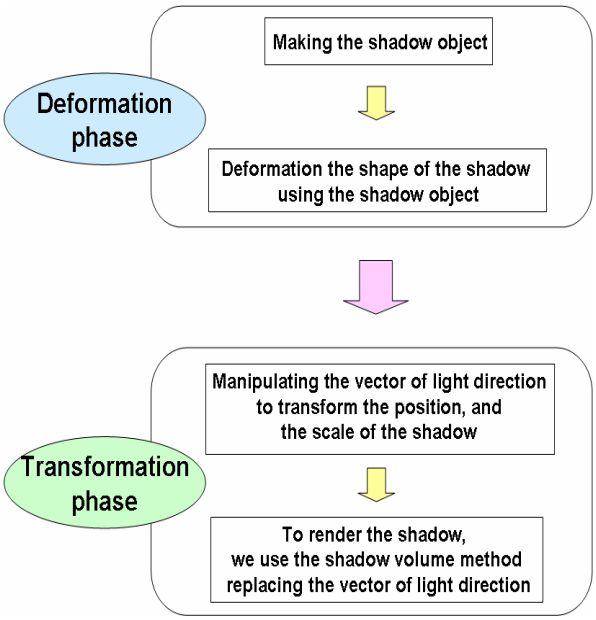


Figure 2. Process Overview

position where the 3D model is. By manipulating the vertex of shadow object directly, we can deform the shape of the shadow. This is called the deformation phase. In the second phase, a shadow is rendered using the shadow volume method. At that time, shadow volume is created by the shadow object instead of the 3D model. Then, animators edit the vector from the light to the 3D model to transform the shadow, such as translation and scaling. This is called transformation phase. An over view of the process is shown in Figure 2.

In section 4, we describe the deformation phase that includes making a shadow object and a deformation technique. In section 5, we describe the transformation phase that includes how to render the shadow and transformation technique.

4. DEFORMATION PHASE

4.1 Making a shadow object

This sub-section explains how to create the shadow object that contributes to the deformation. This object enables animators to edit the shape of the shadow directly. We produce the shadow object by using the shadow projection matrix. The matrix is denoted as $\mathbf{M}$ in the following Equation 1. We calculate $\mathbf{M}$ by solving a ground equation and an equation for a straight line in the direction of a light source. In this case, the ground is the 2-dimension ground that 3D model casts shadow and the line indicates direction of light. The shape of shadow onto the two-dimension ground surface is created using the matrix. The shadow object vertex multiplied by the matrix gives new vertex of the shadow shape projected onto the ground. We set the projected vertex to the position of the 3D model and adjust it to the same scale as the 3D model.

$$\mathbf{M} = \begin{bmatrix} \mathbf{n}\cdot\mathbf{L} - n_x L_x & -n_x L_y & -n_x L_z & 0 \\ -n_y L_x & \mathbf{n}\cdot\mathbf{L} - n_y L_y & -n_y L_z & 0 \\ -n_z L_x & -n_z L_y & \mathbf{n}\cdot\mathbf{L} - n_z L_z & 0 \\ (\mathbf{n}\cdot\mathbf{p}) L_x & (\mathbf{n}\cdot\mathbf{p}) L_y & (\mathbf{n}\cdot\mathbf{p}) L_z & \mathbf{n}\cdot\mathbf{L} \end{bmatrix}, \quad (1)$$

Where, $\mathbf{n} = (n_x, n_y, n_z)$ is the ground's normal vector, $\mathbf{p}$ is a point on the ground, and $\mathbf{L} = (L_x, L_y, L_z)$ is the direction of the light.

4.2 Shadow deformation

This sub-section explains how to control the deformation. We edit a shape of the shadow by using a mouse operation for manipulating each vertex of the shadow object directly in *the editing window*. Note that *the editing window* means the black window on the upper left in figure 3. We call this window *the editing window*. First, animators click on the part of the shadow object where animators want to edit the shape. This region $\Omega[\varepsilon]$ is specified by animators, according to Equation 2. Then, animators drag the part along the direction that they want to

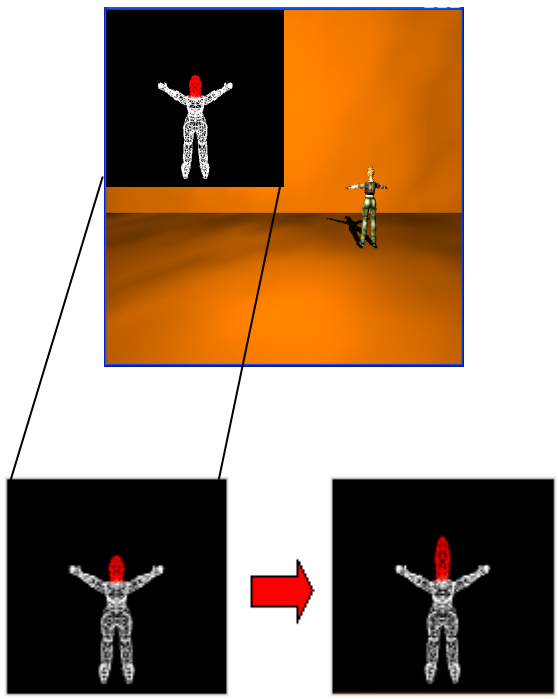


Figure 3. A deformation in *the editing window*. Left image is an original shape of the shadow object. Right image is a shape of shadow object after editing.

 ISBN 978-80-86943-98-5

stretch or compress. In Equation 2, (V_x, V_y) denotes the vertex of a shadow object, and ε is a threshold that decides the region to move. Animators can move vertex (L_x, L_z) to vertex (L'_x, L'_z) of a new shadow object in the region defined by Equation 3 and Equation 4, with (M_x, M_z) being the amount of movement in proportion to a drag of the mouse operation.

$$\Omega[\varepsilon] := \{(Vx, Vy) \mid (Cx - Vx)^2 + (Cy - Vy)^2 < \varepsilon\}, \quad (2)$$

$$L'x = Lx + Mx, \quad (3)$$

$$L'z = Lz + Mz, \quad (4)$$

In addition, we provide the fix mode as an additional mode of deformation phase that gives animators flexibility of editing. In this mode, animators specify the parts that should not move during deformation. Animators set a region by using Equation 2 to fix a vertex with the fix mode. Animators can deform a shadow object outside of the region specified in the fix mode. This mode enables animators to create a shadow object that cannot be achieved by the usual deformation.

Repeating the operation, animators can edit the shape of the shadow that they want to express. Figure 3 shows a deformation example in *the editing window.*

4.3 Animation

Animators can make the animation such as key-frame animation using deformed shadow by our method. Saving the amount of M in Equation 3 and Equation 4 enable animators to create a key-frame animation. For instance, we consider the shadow of

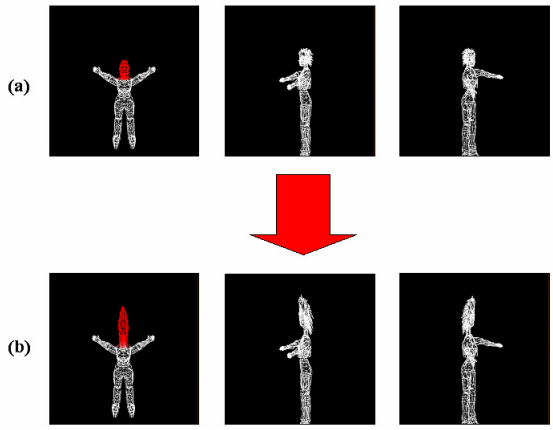


Figure 4. Simple animation example in *the editing window*(Deformation). We deform the shape in the left image, and rotate the 3D model in the middle and right images.

(a) Before editing.

(b) After editing.

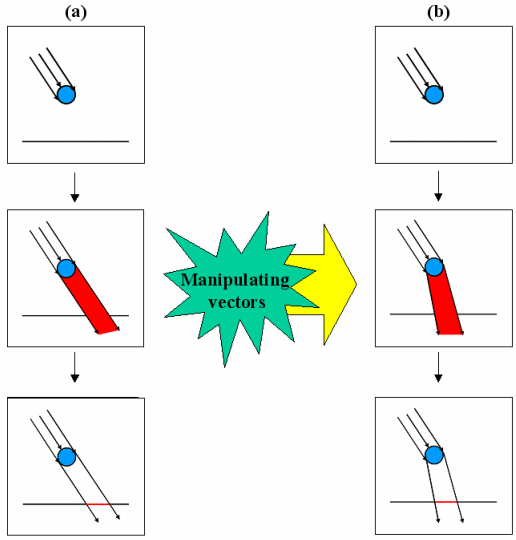


Figure 5. Overview of Transformation. In the top row images, the light illuminates the 3D model. In the middle row images, the shadow volume created using light vectors. In the bottom row images, the shadow on the ground is rendered.

(a) The standard shadow volume steps.

(b) The shadow volume steps after manipulating vectors.

the human character here. The shadow of character's head can be stretched by animators' edit. After editing the shadow, if the 3D model moves freely, the shadow of character's head stretches along the direction that the animators drag. Figure 4 shows this example. These operations are performed *in the editing window*.

5. TRANSFORMATION PHASE

5.1 How to render shadow

To render the shadow onto the ground, we use the standard shadow volume algorithm except replacing the light vector by a per-vertex vector. The shadow volume method is one of the most useful shadowing techniques in 3DCG. Recently, the shadow volume method is mentioned in a lot of research. For instance, [Ass03a] presented the soft shadow using shadow volume. However, soft shadow is not necessarily required for traditional cartoon animation. Therefore, we apply [Eve02a] technique created the umbra wedged shadow, not the penumbra wedged shadow. To define a shadow volume is the space where the light does not illuminate and it is generated by extending vectors from the light source to each outline vertex of the 3D model. In our method, a shadow is transformed by modifying these vectors. Then, the shadow on the ground is produced by the shadow volume using these vectors and the stencil buffer technique [Eve02a]. Figure 5 shows the overview of transformation.

 ISBN 978-80-86943-98-5

5.2 Shadow Transformation: Translation and Scaling

To transform the shadow without changing material information, our method focuses on the extension vectors $\mathbf{L} = (Lx, Ly, Lz)$ and manipulates them. We provide the simple manipulating algorithm below. Equation 5 and Equation 6 represent the translation process. As for Ly, it is not used in this situation, because we edit x-z ground.

$$Lx' = Lx + \alpha, \tag{5}$$

$$Lz' = Lz + \beta, \tag{6}$$

In Equation 5 and Equation 6, α is the extent of the x-direction, β is that of the z-direction, and $\mathbf{L}' = (L'x, L'y, L'z)$ is obtained as a result of translation; α and β are inputted by a simple mouse operation.

Likewise, Equation 7 and Equation 8 represent the scaling process,

$$Lx' = Lx \times \gamma, \tag{7}$$

$$Lz' = Lz \times \gamma, \tag{8}$$

where γ is the extent of scaling decided by a mouse operation, and $\mathbf{L}' = (L'x, L'y, L'z)$ is also obtained as a result of scaling. In these ways, our method can easily achieve the desired transformations.

5.3 Animation

In the similar way of section 4.3, we can create the animation in transformation phase. We obtain the amount of transformation, α, β and γ in Equation 5, Equation 6, Equation 7 and Equation 8 when we transform the shadow object. To save these amounts enable animators to make a key-frame animation. For instance, a shadow is moved from original position to right position by the animators' editing. After editing the shadow, if the 3D model moves freely, position of the shadow moves right compared with the original position. Figure 6 shows a transformation example.

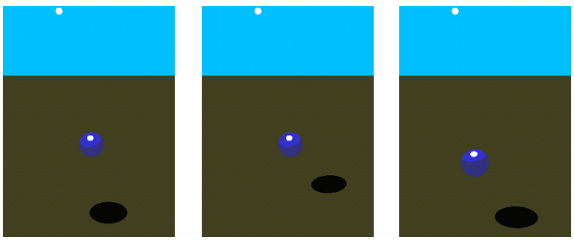


Figure 6. Simple animation example(Transformation). The left image is the original shadow. We transform the shadow in the middle image, and then translate the 3D model in the right image.

6. RESULT

In this section, we demonstrate the editing results created by our tool. The tool runs in real time on an Intel T2600 2.16-GHz platform with NVIDIA GeForce Go, and Directx9.0c as the graphics API.

Figure 7 shows the basic operations provided by our tool. The left image is an original shadow. Starting from original shadow in the most left image, the shadow is first deformed, then translated, and finally scaled. We can control the shadow simply by clicking and dragging with a standard three-button mouse. In usual, animators do not like tedious parameter setting with keyboard. Our method enables animators to create the exaggerating shadow easily by a simple mouse operation.

Figure 1 and Figure 8 demonstrate exaggerating shadow. We make the key-frame animation in the top row of Figure 8. Next, we transform the shadow to exaggerate. We then create the middle row of Figure 8 animation. The shadow in the middle left of Figure 8 is smaller and moves more backward compared to the top left of Figure 8. However, the shadow in the middle right of Figure 8 is bigger and more forward compared to the top right of Figure 8. As a result, the attention of the audience would focus on the shadow. To make more impressive animation, we then deform the shadow. Our method can create the bottom row of Figure 8 from the middle row of Figure 8. This result shows that our method enables animators to spread their expression.

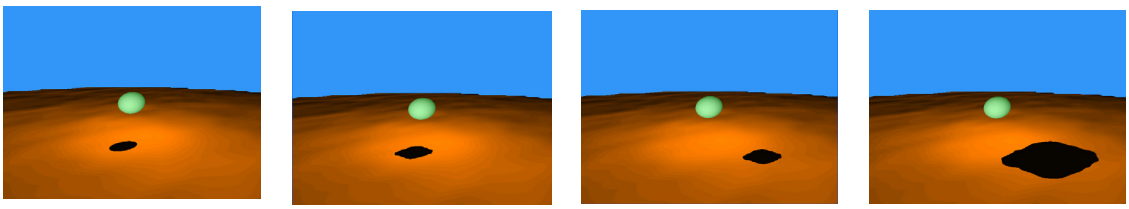


Figure 7. Basic Operation. The left image is the original shadow. Then, we deform, translate and scale the original shadow.

 ISBN 978-80-86943-98-5

Finally, we demonstrate more practical situation using our method. In Figure 9, the shadow plays a dramatic role. The top row of Figure 9 is an original input, and gives the impression of mere shadow. On the other hand, in the bottom row of Figure 9, we exaggerate the shadow to make more dramatic scene in the animation. Traditionally, it is difficult to make the dramatic shadow such as this example, because the procedure for creating 3DCG shadows has a constraint that depends on the relationship between the light source and a 3D model. In that case, animators are required to adjust the light sources or the 3D model to change the shadow. These conventional approaches require animators to perform tedious trial and error in order to obtain an appropriate result. The bottom row of Figure 9 shows that other material information without shadows is same as information before editing the shadow. In this animation, we intended to emphasize a focus on "walking along the wall". Compared to the original animation, the effect is clearly proved by our result. So this also proves that our approach is useful for this kind of practical situation.

7. DISCUSSION AND FUTURE WORK

Unlike photorealistic shadows, the stylized shadows in cartoon animation should not be too detailed nor obeyed by physics-based reality. They should be simpler yet more expressive according to the scenario of the animation. As we mentioned in Section1, The role of shadows is not outstanding expressions in movies or animations such as facial animations or human motions. However, in cartoon animation, animators sometimes would like to use the shadows to foreshadow the successive phenomena such as transforming the 3D characters or objects. These kinds of expressions can be achieved by our approach.

In this paper, we have proposed a novel method of deforming and transforming shadow tweakably

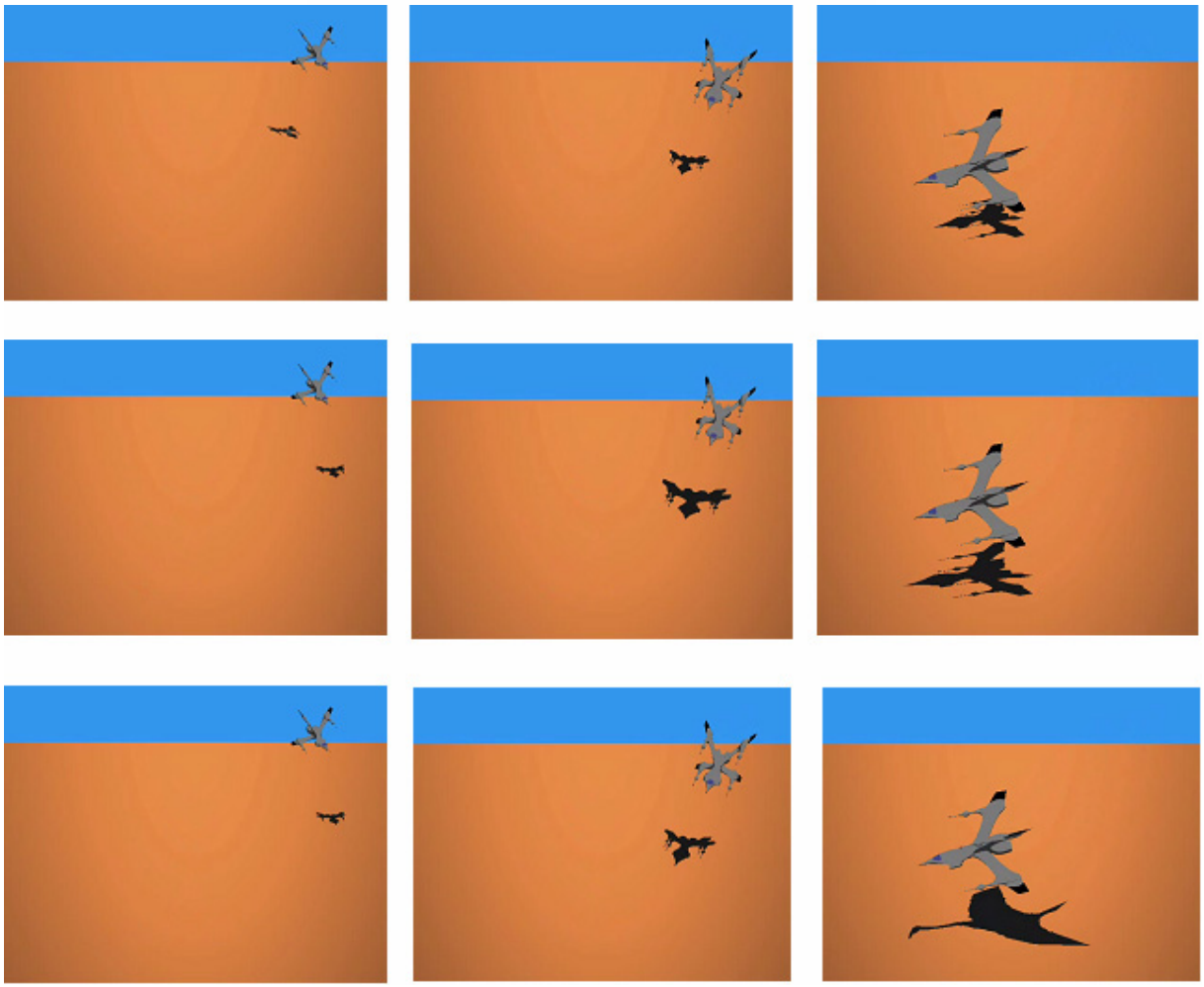


Figure 8. Exaggerating shadow example. The top row images are original key frame animation. The middle row images are transformation of top row animation. The bottom row images are deformation of middle row animation.

 ISBN 978-80-86943-98-5

which allows animators to easily create such stylized shadows and their animations as desired in cartoon animation.

Moreover, our technique could maintain attributes of the edited shadow after transformation or deformation of the shadow object. This result shows that our approach enables animators to create animation intuitively and improves working efficiency.

Of course our final purpose of this research is to express 2D-like shadow by 3D models. Meanwhile our contribution is also expanding the way of expression of 3D shadows.

However, our method has not only several advantages, but also future work. When animators try to rotate the 3D object after deforming the shadow, the shape of shadow occasionally collapses. We improve our method to solve this problem in the future.

This tool is already directable one. However, since animators really need to and like to use an intuitive operation, we should improve our tool to be interactively. For example, animators must deform the shadow in *the editing window* so far. Therefore in order to deform shadows more intuitively, we should deform the shadow directly in the scene. In addition, when we handle easier shapes of shadows or a simple animation, we should develop a more simple method such as a template morphing. That is, given the original shadow and the designed shadow such as a circle, an ellipsoid, and a quadrangle, animators smoothly interpolate the original shadow into the designed shadow in animation. So we can perform animations more simply and quickly.

Treating of self-shadows should also be included in our future direction. As editing shadows, of course we consider how to edit the self-shadows. In addition, we should consider relationship between the shadow itself and its self-shadow.

8. ACKNOWLEDGEMENTS

We would like to thank Ken Anjyo and Aiko Nakatani for their comments and suggestions. This research is supported by Japan Science and Technology Agency, CREST project.

9. REFERENCES

[Kri06a] Krikke. J. Computer Graphics Advances the Art of Anime. IEEE Computer Society, May/ June, pp.14-19, 2006.

[Las87a] Lasseter. J. Principle of Traditional Animation Applied to 3D Computer Animation. Proceedings of SIGGRAPH, pp.35-44, 1987.

[Cor98a] Correa. T. Wagner, Jensen. J. Robert, Thayer. E. Craig, and Finkelstein. Adam. Texture Mapping for Cel Animation, Proceedings of SIGGRAPH, ACM Press, pp.435-446, 1998.

[Rad98a] Rademacher. P. View-Dependent Geometry. In Proceedings of SIGGRAPH, pp.439-

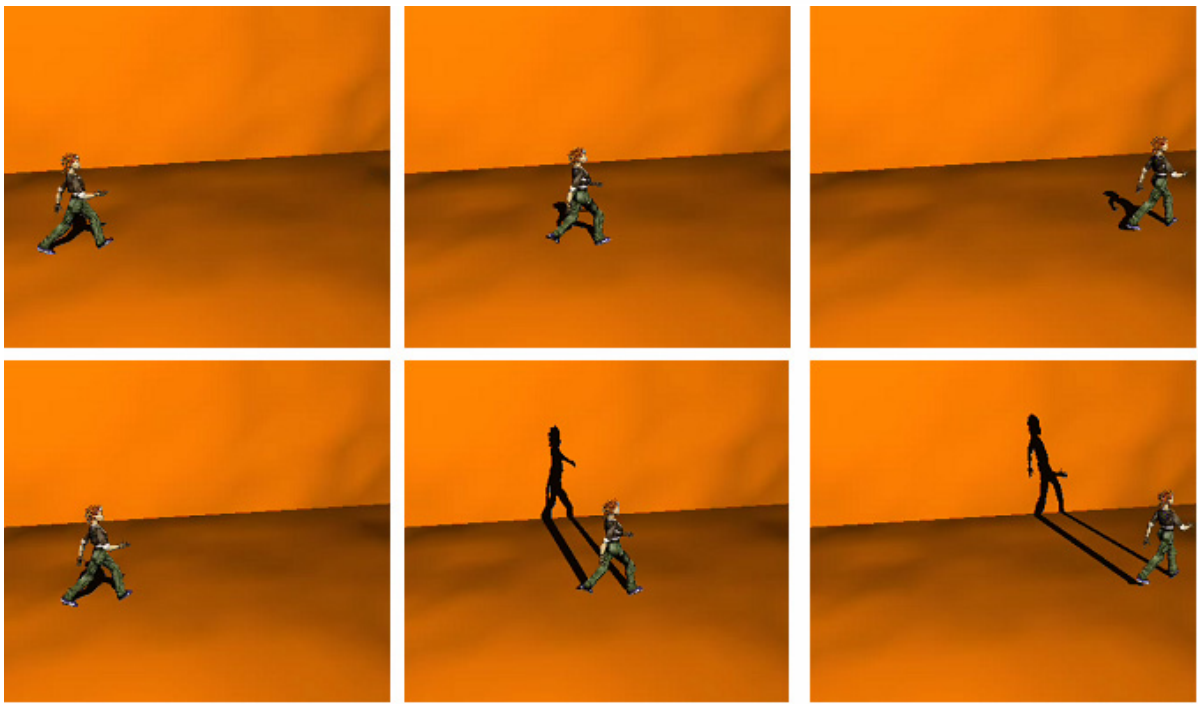


Figure 9. More practical shadow example. The top row images are original key frame animation. The bottom row images are editing of top row animation.

446, 1998.
[Sug06a] Sugisaki. E, Kazama. Y, Morishima. S, Tanaka. N, and Sato. A. Hair motion cloning from cartoon animation sequences. Computer Animation and Virtual Worlds July 2006, vol.17, pp.491-499, 2006.
[Jue06a] Jue Wang, Steven M. Drucker, Maneesh Agrawala, and Michael F. Cohen. The Cartoon Animation Filter. In Proceedings of SIGGRAPH, pp.1169-1173, 2006.
[Chr06a] Christina N. de Juan and Bobby Bodenheimer. Re-using Traditional Animation: Methods for Semi-Automatic Segmentation and Inbetweening. SIGGRAPH/Eurographics Symposium on Computer Animation, pp.223-232, 2006.
[Bar97a] Barzel. R. Lighting control for computer cinematography. Journal of Graphics Tools 2, pp.1-20, 1997.
[Pet00a] Petrovic. L, Fujito. B, Williams. L, and Finkelstein. A. Shadows for cel animation. In Proceedings of SIGGRAPH2000, pp.511-516, 2000.
[Pel02a] Pellacini. F, Tole. P, and Greenberg. D. A user interface for interactive cinematic shadow design. ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH2002 21, pp.563-566, 2002.
[Lak00a] Lake.A, Marshall.C, Harris. M, and Blackstein. M. Stylized rendering techniques for scalable real-time 3D animation. Proceedings of Non-Photorealistic Animation and Rendering 2000, pp.13-20,2000.
[Anj03a] Anjyo.K and Hiramatsu. K. Stylized highlights for cartoon rendering and animation. IEEE Computer Graphics and Applications 23, pp.54-61, 2003.
[Anj06a] Anjyo. K, Wemler. S, and Baxter. V. W. Tweakable Light and Shade for Cartoon Animation. In Proc. NPAR 2006, pp.133-139, 2006.
[Ass03a] Assarsson. U, Akenine-moller. T. A Geometry-Based Soft Shadow Volume Algorithm using Graphics Hardware. ACM Trans, pp.511-520, 2003.
[Eve02a] Everitt. C, and Kilgard. M. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. http://developer.nvidia.com, 2002.

Assembly Simulation on Collaborative Haptic Virtual Environments

R. Iglesias, E. Prada
Fundación Labein - Tecnalia
Parque Tecnológico Zamudio
Edificio 700
Spain E-48160, Derio
{riglesias,eprada}@labein.es

A. Uribe, A.Garcia-Alonso
U. of the Basque Country
Facultad Informática
Manuel de Lardizabal 1
Spain E-20018, San Sebastián
alex.galonso@ehu.es

S. Casado, T. Gutierrez
Fundación Labein - Tecnalia
Parque Tecnológico Zamudio
Edificio 700
Spain E-48160, Derio
{sara, tere}@labein.es

ABSTRACT

Currently, Virtual Environments (VEs) are used within engineering industry: physical prototypes or mock-ups are replaced by virtual prototypes. Increasingly, these VEs also allow designers and engineers to carry out assembly/disassembly processes and assembly sequences before any physical prototype is built. Moreover, different designers or engineers, which may be situated in the same or in geographically dispersed locations, often collaborate in the design of products. This allows developing more complex products within a shorter time scale and lowered costs. On the other hand, the utilization of haptic feedback has been found to significantly enhance task performance, for instance, in assembly tasks. In this paper we describe an assembly simulation application on a collaborative haptic virtual environment, where several users interact with virtual models to perform assembly operations within the same virtual scene. The paper also summarizes results achieved with experiments which evaluated different collaborative architectures. Furthermore, it reports on the goals that can be achieved and the limitations for haptic collaborative interaction in each case.

Keywords

Virtual reality; Collaborative virtual environment; Virtual prototyping; Haptic feedback; Assembly simulation.

1. INTRODUCTION

The traditional design systems (CAD, CAM and CAE) allow generating 3D designs and simulating the behavior of the product and part of the manufacturing process, such as assembly/disassembly (A/D) processes and sequences for training. However, they do not integrate all the physical processes of the real world. The use of haptic devices (sense of touch) is a powerful technology that can enhance and solve some of the limitations of the traditional simulation systems.

Haptic devices are generally used to indicate a class of mechanical system that is intended to replicate forces and local continuous stimuli on specific areas of human body: finger, hand or body. Today, several haptic devices with different specifications can be available, such as, PHANToM Premium (Figure 1), PHANToM Omni and GRAB. PHANToM is developed and distributed by SensAble Technologies (Cambridge MA, USA). GRAB, which provides a larger workspace and two points of contact, was developed by PERCRO (Scuola Superiore Sant'Anna, Italy).

These days, haptic technology is an emerging field, that is being successfully applied to a wide range of applications, for instance, training simulators [Bas01a], visually impaired people applications [Igl04a], entertainment and gaming [Zho04a], as well as industrial design and maintenance [Bor04a], [Pet04a], [Igl06a]. In this latter application, the utilization of haptic devices has been found to significantly improve operation effectiveness in assembly tasks [Bas00a], [Sal00a], [Pet04a].

On the other hand, nowadays, products are increasingly being developed by geographically dispersed design teams. These may be located in different partner companies, or different offices of the same company, perhaps even in different countries. On large projects, different design teams meet regularly for preliminary reviews, design reviews, defect reviews and so on. It is becoming

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

strategically important to be able to link distributed design teams, to permit concurrent access and modification of design models from different geographical locations.

There are different types of distributed collaborative virtual environments (CVEs), for instance, some provide collaborative visualization while, others, not so many, also consider haptic interaction. The developments of Hagsand [Hag96a], Borro [Bor00a] and Greenhalgh [Gre00a] are focused on distributed visualization: a user interacts with a virtual model while others can watch it. Other examples about distributed VEs can be found in Mclaughlin [Mcl02a] and Burdea [Bur03a].

Figure 1. A user interacting with PHANToM Premium.

The work presented in this paper provides a helpful tool for the design and maintenance teams. The designers would be able to check the validity of the design, simulate A/D processes in order to simplify the process, avoid interferences and define A/D procedures with haptic interaction. The procedures defined by the design team could also be the base for training operators in new or complex assembly and maintenance (A/M) tasks.

This paper describes an application to simulate assembly operations on a collaborative virtual environment (CVE). This application allows different users to analyze in real-time new products, A/M operations without using physical models, by means of realistic navigation, visualization and interaction with the virtual models by means of traditional devices (i.e. keyboard, mouse) or haptic devices.

Sections 2 and 3 explain the toolkits in which the assembly application is based on. The main challenges on collaborative haptic virtual environment are presented in Section 4. It also includes one example and the results that lead to the conclusions.

2. TOOLKITS: *DAT*um AND ASSEMBLY SIMULATOR

The research described later on in this paper requires the integration of three toolkits: ***DAT***um, an *Assembly Simulator* and a *Haptic Assembly Simulator*. The next section deals with the third toolkit. The three toolkits described between this section and the following one allows analyzing new products in real-time and simulating virtual A/M operations via keyboard, mouse or haptic devices.

DATum is an object oriented variational non-manifold geometric modeler developed by LABEIN, with a STEP translator compliant with ISO 10303-AP203 (International Standard for the representation and the exchange of product data between different CAD systems).

DATum uses a hybrid representation scheme between the two most common representations within the field of the Geometric Modeling: the Constructive Solid Geometry [Req77a] and the Boundary Representation [Man84a], exploiting the advantages of each one of these representations. In this way, a model can be created through Boolean operations (union, intersection and difference) between other two models, and it has always associated a boundary representation. The boundary representation of a model provides both geometric and topological information. In relation to the geometry, ***DAT***um supports both basic geometry (conics and quadrics) and complex geometry (NURBS curves and surfaces). Its topological structure is based on the Weiler structure [Wei86a].

DATum is a non-manifold modeler [Wei86a]. This capability allows to represent solid, surface and wireframe models in a unified and simultaneous way and to deal with the “region” concept. In this way, a model can be composed by several regions associated, for example, to different materials.

It is also a variational modeler. The variational geometry is based on the definition and modification of geometric models through a set of functional restrictions, instead of the classical parameters. In this way, the model can include the design intent.

On top of ***DAT***um, LABEIN has developed an Assembly Simulator that combines direct manipulation techniques, collision detection, automatic assembly constraint recognition and management within a unified framework to allow undertaking maintenance operations of mechanical assemblies interactively. It consists mainly of two modules: collision detection and assembly constraint recognition.

 ISBN 978-80-86943-98-5

The collision detection module detects any collision between an object that is being moved by the user (through translations and/or rotations) and any other object of the working space. In this way, the object can not penetrate into another and only real feasible movements are allowed. The implemented algorithm is based on the 'RAPID' library implemented by Gottschalk [Got96a]. This is a polygon interference detection library for large environments. The degree of performance of this algorithm is quite related to the quality of the triangles, so the more regular the triangles are a higher performance will be achieved. Algorithms to get suitable triangles were developed using the basic functionality of ***DAT***um.

When a collision is detected the user is not allowed to move in the collision direction, but he must change the movement direction in order to avoid the collision. Sound aids and changes of colors have been implemented to warn the user when a collision is produced.

The *Assembly Simulation* module allows the automatic recognition of the potential constraints between a model that is being moved by the user and the rest of the components of the mechanical assembly. Once the system detects a constraint, the movement of the model will be constrained to satisfy the active constraint detecting also the collisions with the other model of the workspace [Gut98a], [Bar98a], [Bar99a]. This module makes use of the collision detection algorithms explained above and it is based on the information about the adjacency relationships of the topological entities of the models, for example how the faces are connected to detect if a face generates a hole or a protrusion. The following A/D constraint methods are depicted in Figure 2: objects along a common axis (pin-hole), and objects along coincident planar faces. Other constraints are pin-multiple holes, hole-pin, pin-pin and hole-hole.

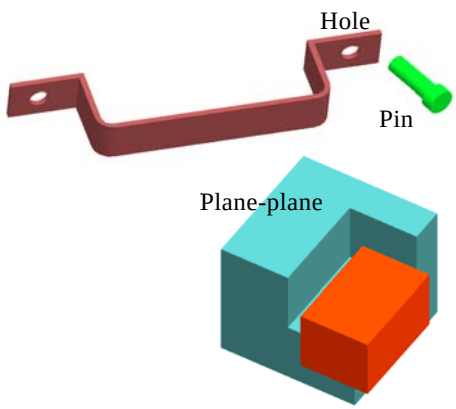


Figure 2. Assembly/Disassembly: pin-hole and plane-plane.

3. TOOLKITS: HAPTIC ASSEMBLY SIMULATOR

The integration of a haptic device within ***DAT***um allows the user to interact with 3D designs in a new and more realistic way than the traditional systems. The user can not only view the objects designed, but also interact with them: touching, grasping and moving them within the virtual scene, detecting and feeling the possible collisions and assemblies among models. Experiments used different haptic devices: the PHANToM device, PHANToM Omni and GRAB device (see Section 1). Haptic devices were used for the following manipulation tasks: touch, move and collide, and Assembly/Disassembly operations.

The user can touch any 3D model and move itself along its external surface, detecting its edges and corners. The algorithm to touch and interact with a virtual object by means of a haptic device is based on the analysis of the position of the user's finger (recovered by the haptic device) with respect to the object to check if the point is inside or outside of the object. In this case, the force sent by haptic device will be proportional to the penetration depth of the user's finger into the virtual object and normal to the object surface.

Any object, that can be touched, can also be *grasped and moved* (through translations and rotations, called transformations in general) by the user along the virtual workspace.

This utility also detects any collision between the object that is being moved and any other object of the workspace. A model cannot penetrate into another and only real feasible movements are allowed.

The workflow repeats the following steps:

1) Calculate the transformation described by the movement of the user's finger (movement of the end-effector of the haptic device): translation and/or rotation.

2) Study if, in the new position, the object is colliding with any other object of the workspace.

3) If there is not collision, apply the transformation of the object. Calculate the force to be sent to the user, depending on the result of the collision detection. In order to provide a more realistic interaction, the object weight has also been implemented. This has required the implementation of some algorithms to calculate the volume and the area of any 3D object. If there is not any collision, the force sent by the haptic device corresponds to the object weight. Whereas if the object is colliding, a force

 ISBN 978-80-86943-98-5

opposing to the movement direction to come back to the last position.

Assembly/Disassembly allows simulating A/M operations of a mechanical assembly and it is based on the automatic recognition of constraints provided by the Assembly Simulator explained above. Most of the previous utilities (touch, move, detect collisions) are enclosed here.

The workflow repeats the following steps:

1) Calculate the transformation described by the movement of the user's finger (movement of the end-effector of the haptic device): translation and/or rotation.

2) This transformation is sent to the *Assembly Simulator* to study if, in the new position, the object is colliding with any other object of the workspace, breaking an existing constraint or satisfying a new one.

3) The *Assembly Simulator* calculates the adequate movement to be applied to the object and information about the constraints that are satisfied. With this information the force to be sent to the user is calculated.

Three types of sliding forces were implemented to support assembly methods: on a surface, on a line and on a point.

1) *Sliding forces on a surface* allow two objects to remain in contact along a common surface. An external force can compel the user to move his finger, and so the object fixed to it, on a given surface. The user can move freely along the surface but she can not move away from it (unless the user exerts an incremental force to break the active constraint). The algorithm to calculate this force is based on the computation of the point of minimum distance from a point to a surface. In this case, the force corresponds to the vector defined by the user's position and the point of minimum distance on the surface.

2) *Sliding forces on a line* allow two objects to remain in contact along a common line. As in the previous case, the user can move his finger, and so the object fixed to it, on a given line. The user can move freely along the line but he can not move away from it (unless the user exerts an incremental force). The algorithm to calculate this force is based on the computation of the point of minimum distance from a point to a line. In this case, the force corresponds to the vector defined by the user's position and the point of minimum distance on the line.

3) *Sliding forces on a point* allow two objects to remain in contact along a common point. With this type of force, the user can only rotate the object fixed to her finger about its own axis, unless the user exerts an incremental force. In this case, the force corresponds to the vector defined by the user's position and the point.

4. CVEs FOR ASSEMBLY SIMULATION

There are different types of *distributed collaborative virtual environments* (CVEs), for instance, some only provide collaborative visualization whereas, others, not so many, also consider haptic interaction. This section analyses the problem of assembly simulation on CVEs where users can simultaneously interact within the same scene using traditional or haptic devices. First, CVEs are considered, and then a practical case is analyzed. The last subsection summarizes different system architectures that were experimented extending the toolkits described in sections 2 and 3.

4.1 Collaborative assembly application

A CVE implies a distributed system that allows geographically separated users (computers) to communicate and/or interact within the same virtual scene through connected networks such as, LAN or the Internet.

In our application, mechanical assemblies can be designed within ***DAT***um, or imported from another CAD system through STEP files.

Each client can interact within the virtual scene, either moving freely and touching models or grasping a model. In this latter case the user feels collisions with other models along their movement and the system may guide the user to undertake an assembly method. Users can interact with the virtual scene using different devices: mouse, keyboard or haptic devices (see Section 1).

Each client replicates the same virtual scene, managing their visualization, from a different point of view. It is not necessary that all users connect themselves simultaneously to the work session. A user can be joined the work group when it is considered opportune; when a new user connects to the server, this user will receive the state of the environment upon connection.

Several clients may undertake different actions on the same CVE. For instance, one client can freely move, while other can move a model satisfying a constraint (assembly), and another different client can collide with a different object. During these tasks, *consistency*, that is virtual scene synchronization, must be guaranteed for all clients.

 ISBN 978-80-86943-98-5

Whenever a user tries to move the grasped object, the system must validate the movement considering potential collisions with the rest of the objects within the scene. Constraints must also be considered depending on the type of assembly method chosen by the client. It is important the way in which communications, data and processes are distributed between server and clients to provide a realistic interaction among users and to compute immediate responses as in the case of haptic devices with high frequencies.

Forces applied to the user by the haptic device must provide an adequate sensation. This topic has been deeply addressed by other authors [Bur03a].

User interaction on a distributed assembly application must be similar to the standalone (not networked). However, some restrictions necessarily appear, i.e. opening/closing virtual environments, an grasped by one user can not be simultaneously grasped by another user.

Problems regarding network communications also play an important role in the design of specific implementations. However, the inevitable conditions of the network (e.g. latency, jitter, background traffic) affect distributed applications in a different way as it is considered in the next subsection.

4.2 Practical case

This subsection describes a specific experiment, where the virtual scene was an aeronautical assembly (Figure 3) provided by an engineering company, SENER.

This CVE allowed the simulation of assembly tasks with simultaneous interaction of several users. This was achieved by adopting a client-server architecture, as Figure 4 shows. The distributed components were developed with ORBacus (an Object Request Broker that is compliant with CORBA specification).

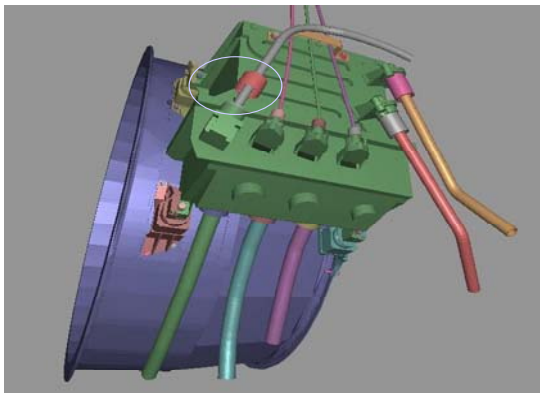


Figure 3. Aeronautical assembly (an electrical box for an aircraft engine) provided by SENER.

A server administrates all data received from clients: new virtual scene, request of grasping a model, request of transformation of a model, new client and so on. The server manages the selection of models to avoid that two different users simultaneously grasp the same model.

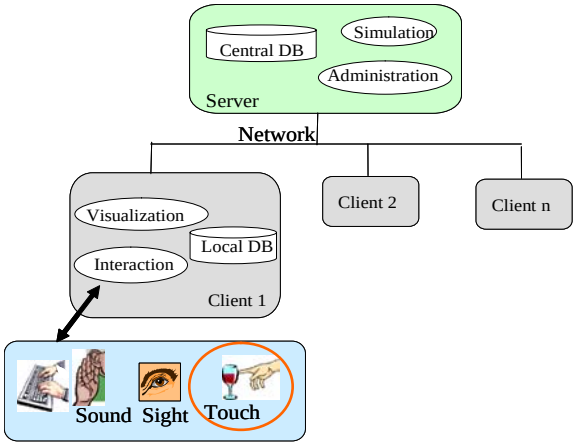


Figure 4. Client-Server architecture.

The original assembly was done in Pro-E and read into ***DAT***um through STEP. A distributed session was run between two users through a local area network simulating the assembly process. A user could interact using a haptic device: grasping a model, feeling its weight, detecting the collisions with other objects along their movement and simulating the assembly methods explained above. Meanwhile, the other user could interact with the shared scene with the keyboard.

Previously, with this design an assembly problem was found [Car03a]. During the assembly path a collision with the green box does not allow to finish the assembly process (Figure 3). Therefore, a re-design was needed in order to avoid this problem.

Using this architecture the application worked properly and consistency between clients was guaranteed all the time.

4.3 Architectures for distributed environments

Some research has been done on CVEs where virtual scene synchronization (consistency), effective and compelling haptic feedback (quality of force feedback) and scalability continue to be enormous challenges. There are different architectures to support distributed systems: peer-to-peer [Cla01a], client-server [Sin99a] or a mixture of them [Bor00], [Mar06a]. Marsh et al. [Mar06a] analyze different architectures supporting haptic interaction on CVEs and provide an updated review of the research performed to deal with the challenges described in the Section 4.1.

 ISBN 978-80-86943-98-5

Iglesias et al. [Igl05a] compare three different client-server architectures for assembly simulation on a CVE described in this paper. That research studied how could be distributed the simulation workload between server and clients. Two architectures provided interesting results. In architecture A3 clients deal with the validation of movement and consistency maintenance is managed at server side. On the other hand, in architecture A1 consistency is trivially achieved because server sequentially validates all movements. When there are more than two users, the architecture A3 had a better performance.

With client-server architectures consistency is easily achieved, it is calculated once either at server side or trivially guaranteed if server validates all movements. No complex synchronization mechanism is required. However, in such architectures, haptic interaction may specially be affected due mainly to network conditions (i.e. delay, jitter). This plays an important role in situations where force feedback at a user depends on the actions of another user (*dependent interaction*). This case happens when two users grasp a different object and for instance, a user assemble the grasped object into another one being held by a different user (*remote assembly*). Not only performing an assembly, but when both grasped objects collide and users should feel the corresponding collision force feedback (*dependent collision*).

As a consequence, these client-server architectures limit the use of haptic interaction to extremely good network conditions between the client and server. If that previous condition is not achieved, users should avoid working in the vicinity of objects grasped by other users. The haptic interaction may be affected in case of dependent interaction. Marsh et al. [Mar06a], which used the same assembly application proposed in this paper, reported on a hybrid architecture that only supports haptic interaction for simultaneous cooperative haptic tasks over a low delay network (between the client and server) and in other case, users collaborate by taking turns.

As a result of the impact of network conditions on haptic interaction with client-server architectures, in Iglesias [Igl06a] a peer-to-peer architecture is presented. This paper aims to achieve a higher degree of collaboration, at least between two users: to achieve nearby collaboration, such as, to carry out remote assemblies and maintain consistency even when network conditions get worse. A new consistency-maintenance scheme was proved to maintain consistency. Results were satisfactory with different network conditions.

5. CONCLUSIONS

In recent years, VEs are used within engineering industry: physical mock-ups are being replaced by virtual prototypes. Increasingly, these VEs allow designers and engineers to carry out assembly/disassembly processes before any physical prototype is built. On the other hand, several users often collaborate in the design of a product, evaluating assembly sequences, which may be situated in the same or in geographically dispersed locations.

Moreover, the utilization of haptic devices allows users to have physical interaction with digital mock-ups. And the haptic feedback (sense of touch) has been found to significantly enhance task performance, such as these assembly tasks.

We describe an application to carry out assembly operations on a collaborative virtual environment with the use of keyboard, mouse or haptic devices. The Haptic Assembly Simulator recognizes automatically collisions, assembly constraints and replicates properly forces on user's fingers to provide an effective interaction.

Section 4.3 shows which collaborative interaction goals can be achieved using different network topology architectures and strategies. Client-server architectures provide good results if network conditions are good enough and objects managed by users are sufficiently separated. A peer-to-peer architecture has been proposed in order to support a collaborative assembly task with certain network delay. With this architecture, a remote assembly (a user assembles an object into another object grasped by a different user) can be performed even with the worsening of network conditions. A consequence is that, although there is not a global solution to the problem yet, different network topologies may be adopted to build applications with specific goals.

6. ACKNOWLEDGMENTS

This work has been partly supported by ENACTIVE Interface, European project IST-2002-002114; Gipuzkoako Foru Aldundia project (13/2005) with FEDER funds; Spanish Ministry of Education and Science, grant TIN2006-14968-C02-01; and AMIGUNE Basque Government project (2005-2007).

7. REFERENCES

[Bar98a] Barbero, J.I., Gutiérrez, T. and Eguidazu, A., Haptic Virtual Prototypes for Assembly Simulation, in ISATA Conference No.31, pp. 109-117, 1998.

[Bar99a] Barbero, J.I., Gutiérrez, T., Alvarez, A. and Carrillo, A., Assembly Simulation Tools:

Tolerance Analysis and Haptic Virtual Environment, in EAEC European Automotive Congress, 1999.

[Bas00a] Basdogan, C., Ho, C.H., Srinivasan, M. A. and Slater,M., An Experimental Study on the Role of Touch in Shared Virtual Environments. ACM Transactions on Computer-Human Interactions, 7, pp. 443-460, 2000.

[Bas01a] Basdogan, C., Ho, C., Srinivasan, M.A., Virtual Environments for Medical Training: Graphical and Haptic Simulation of Common Bile Duct Exploration. IEEE/ASME Transactions on Mechatronics, 6, pp. 267-285., 2001.

[Bor00a] Borro, D., Matey, L., Sánchez, H., Recio, I., Garcia-Alonso, A., CSCW for foundry design using Java 3D, demonstration at ACM 2000 Conference on Computer Supported Cooperative Work, 2000.

[Bor04a] Borro, D., Savall, J., Amundarain, A., Gil, J.J., García-Alonso, A., and Matey, L., Large Haptic Device for Aircraft Engine Maintainability. IEEE Computer Graphics and Applications, 24, 70-74, 2004.

[Bur03a] Burdea, G.C., and Coiffet, P., Virtual Reality Technology, John Wiley and Sons, 2003.

[Car03a] Carrillo, A.R., Beloki, O., Casado, S., Gutierrez, T., Barbero, J.I., Virtual Assembly and Disassembly Simulation on a Distributed Environment, in Proceedings of Virtual Concept, 2003.

[Cla01a] Clark, D., Face-to-Face with Peer-to-Peer Networking. Computer, Vol. 34, No. 1, pp. 18-21, 2001.

[Got96a] Gottschalk, S., Lin, M.C. and Manocha, D., OBBTree: A Hierarchical Structure for Rapid Interference Detection, in Proc. of ACM Siggraph, pp. 171-180,1996.

[Gre00a] Greenhalgh, C., Purbrick, J., and Snowdon, D., Inside MASSIVE_3: Flexible support for data consistency and world structuring, in ACM Conference on Collaborative Virtual Environments, pp. 119-127, 2000.

[Gut98a] Gutiérrez, T., Barbero, J.I and Eguidazu, A., Virtual Assembly and Disassembly, in IFAC workshop on intelligent Assembly and Disassembly, pp. 35-40, 1998.

[Hag96a] Hagsand, O., Interactive Multiuser VEs in the DIVE System. IEEE Multimedia, 3, pp. 30-39, 1996.

[Igl04a] Iglesias, R., Casado, S., Gutiérrez, T., Barbero, J.I., Avizzano, C.A., Marcheschi, S., Bergamasco, M., Computer graphics access for blind people through a haptic and audio virtual environment, in IEEE International Workshop on Haptic Audio Visual Environments and their Applications, pp. 13-18 , 2004.

[Igl05a] Iglesias R., Casado S., Gutierrez T., Barbero J.I. , Sanchez E., Garcia-Alonso A. “Architectures for distributed multimodal virtual environments”, International Conference on Manufacturing Research, England (2005a).

[Igl06a] Iglesias, R., Carrillo, A.., Casado, S., Gutiérrez T. and Barbero, J.I., Virtual Assembly Simulation in a Distributed Haptic Virtual Environment, in International conference on Advanced Design and Manufacture, 2006.

[Igl06b] Iglesias R., Casado S., Gutiérrez T., García-Alonso A., Meng K., Yu W. and Marshall A. “A Peer-to-peer Architecture for Collaborative Haptic Assembly”, Proceedings of the 10-th IEEE International Symposium on Distributed Simulation and Real Time Applications, pp. 25-34, 2006.

[Man84a] Mantyla, M., A Note on the Modelling space of Euler operators. Computer Vision, Graphics and Image Processing, 26, 45-60, 1984.

[Mar06a] Marsh J., Glencross M., Pettifer S. and Hubbold R., “A network architecture supporting consistent rich behaviour in collaborative interactive applications”, IEEE Transactions on visualization and computer graphics, Vol. 12, No. 3, pp. 405-416, May 2006.

[Mcl02a] Mclaughlin, M.L., Hespanha, J.P., Sukhatme, G.S., Touch in Virtual Environments: Haptics and the Design of Interactive Systems, Prentice Hall, 2002.

[Pet04a] Petzold, B., Zaeh, M.F., Faerber, B., Deml, B., Egermeier H., Schilp J. and Clarke, S., A Study on Visual, Auditory, and Haptic Feedback for Assembly Tasks. Presence: Teleoperators & Virtual Environments, 13, pp. 16-21, 2004.

[Req77a] Requicha, A.A.G. and Voelcker, H.B., Constructive Solid Geometry. Tech. Memo 25, Production Automation Project, University of Rochester, 1977.

[Sal00a] Sallnäs, E.L., Supporting Collaboration in Distributed Environments by Haptic Force Feedback. ACM Transactions on Computer-Human Interaction, 7, pp. 461-476, 2000.

[Sin99a] Singhal, S. and Zyda, M., Networked Virtual Environments Design and Implementation, ACM Press SIGGRAPH Series, Addison Wesley, 1999.

[Wei86a] Weiler, K., Topological structures for Geometric Modeling. PhD thesis, Dept. Comput. Syst. Engr., Rensselaer Polytechnic Institute, 1986.

[Zho04a] Zhou, S., Wentong, C., Lee, B. and Turner, S. J., Time-Space Consistency in Large-Scale Distributed Virtual Environments. ACM Transactions on Modeling and Computer Simulation, 14, pp. 31-47, 2004.

Viewpoint Entropy-Driven Simplification

P. Castelló
Universitat Jaume I
Campus Riu Sec
E-12071, Castelló, Spain
castellp@lsi.uji.es

M. Sbert
Universitat de Girona
Campus Montilivi
E-17071, Girona, Spain
mateu@ima.udg.es

M. Chover
Universitat Jaume I
Campus Riu Sec
E-12071, Castelló, Spain
chover@lsi.uji.es

M. Feixas
Universitat de Girona
Campus Montilivi
E-17071, Girona, Spain
feixas@ima.udg.es

ABSTRACT

In this paper, a new viewpoint-based simplification approach is proposed for polygonal meshes. This approach is driven by an information-theoretic metric, viewpoint entropy, which measures the amount of information from a scene or object that arrives at a certain viewpoint. Our algorithm applies the best half-edge collapse as a decimation criterion and uses the variation of viewpoint entropy to measure the edge collapse error. Compared to pure geometric-based simplifications, the models produced by our method are closer to the original model according to visual similarity. Our approach also achieves a higher simplification in hidden interiors, by being able to remove them all and to leave the visible areas of the mesh intact. Models generated by CAD applications can benefit from this feature, since these models are usually constructed by assembling smaller objects which can become partially hidden during joining operations. The main application of our method is for video games where models come from CAD applications and are geometrically not very complex, a few thousand polygons at the most, and in which visual similarity is the most important requirement.

Keywords

Simplification, level-of-detail, viewpoint selection, Information Theory.

1. INTRODUCTION

Most common simplification methods use some technique based on a geometric distance as a quality measure between an original mesh and the one obtained from simplification. With these methods we can achieve meshes that are very similar to the original. On the other hand, image-based simplification methods carry out a simplification guided by differences between images more than by geometric distances. That is, their goal is to create simplified meshes that appear similar according to visual criteria. Thus, the applications that can benefit from image-based methods are those in which the main requirement is visual similarity. Examples of such applications are video games, vehicle simulations, walk-throughs, etc. A reduced number of applications, however, require exact geometric tolerances with regard to the original model. For this type of applications it would be better to consider some simplification method based on a pure geometric measure. Examples of such applications include collision detection and path planning for part insertion and removal.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

Geometric methods are suitable for scanned models, which are composed of thousands of polygons. However, in video games models usually come from CAD applications and it is very useful to simplify models that are not very complex to a lower level, typically a few hundred polygons. This is where our approach could be taken into consideration.

In this paper, we introduce a new viewpoint-based simplification method which uses viewpoint entropy [Vaz01], a measure of the geometric information of a scene or object seen from a certain point of view. This method uses the best half-edge collapse as a decimation criterion and measures the variation of viewpoint entropy to quantify the cost of collapse. Experimental results show our method yields better visual performance than QSlim-based simplifications [Gar97]. Our algorithm also offers very good results even at early stages of simplification, where achieves a higher simplification in hidden interiors.

This paper is organized as follows. In Section 2, we survey the previous work and basic information-theoretic measures. In Section 3, we define the simplification error metric based on viewpoint

entropy to measure the cost of an edge collapse. In Section 4, we describe our simplification algorithm. In Section 5, we show the results of our experiments and finally, in Section 6, conclusions and future work are presented.

2. BACKGROUND

In this section we review related work, basic information-theoretic measures and information-theoretic viewpoint selection measures.

Related Work

The most important improvement in geometry-oriented simplification methods in recent years was the incorporation of mesh attributes such as color, normals and textures. For example, Hoppe extended his initial work [Hop96] by proposing a new quadric metric that incorporates colors and texture coordinates [Hop99], and the QSlim algorithm [Gar97] was also extended with those attributes in [Gar98]. Cohen et al. [Coh98] developed an algorithm based on edge collapses that samples the vertex position, normal and color attributes of the original mesh and then converts them to normal and texture maps. This algorithm is based on a texture deviation metric.

Lindstrom et al. [Lin00] addressed the problem of visual similarity by developing a pure image-based metric. Basically, their method determines the cost of an edge collapse operation by rendering the model from several viewpoints. The algorithm compares the rendered images to the original ones and adds the mean-square error in luminance across all pixels of all images. Then it sorts all edges by the total error induced in the images and chooses the edge collapse that produces the least error. They used 20 viewpoints in their implementation to compute that error. The main advantage of their method is that the metric provides a natural way to balance the geometric and shading properties without requiring the user to perform an arbitrary weighting of these attributes. On the other hand, its main disadvantage is the low speed it achieves.

Luebke et al. [Lue01] presented a method to perform a view-dependent polygonal simplification using perceptual metrics. These metrics derive from a measure of low-level perceptibility of visual stimuli in humans. Later on, Williams et al. [Wil03] extended this work for lit and textured meshes. Zhang et al.[Zha02] proposed a new algorithm that takes visibility into account. Their approach defined a visibility function between the surfaces of a model and a surrounding sphere of cameras. The number of cameras increases both accuracy and calculation time. They used up to 258 cameras. In order to guide the simplification process, they combined their visibility measure with the quadric measure introduced by Garland et al. [Gar97].

Recently, Lee et al. [Lee05] introduced the idea of mesh saliency as a measure of regional importance for graphics meshes. This measure was incorporated into mesh simplification. Basically, their approach consists in generating a saliency map, and then simplifying by using this map in the QSlim algorithm as in [Zha02]. The new edge collapse cost is that of the quadric times the saliency of this edge.

Information-Theoretic Measures

Let X be a finite set, let X be a random variable taking values x in X with distribution $p(x)=Pr[X=x]$. Likewise, let Y be a random variable taking values y in Y. The *Shannon entropy H(X)* of a random variable X is defined by

$$H(X) = -\sum_{x \in X} p(x) \log p(x). \tag{1}$$

This is also denoted by $H(p)$ and measures the average uncertainty of a random variable X. All logarithms are base 2 and entropy is expressed in bits. The convention that $0 \log 0 = 0$ is used.

The *mutual information* (*MI*) between X and Y is defined by

$$I(X,Y) = \sum_{x \in X} p(x) \sum_{y \in Y} p(y \mid x) \log \frac{p(y \mid x)}{p(y)}. \tag{2}$$

This is a measure of the information shared by X and Y. It can be seen that $I(X,Y)=I(Y,X)\geq 0$.

The *relative entropy* or *Kullback-Leibler distance* between two probability distributions p and q defined over the same set is given by

$$KL(p \mid q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}, \tag{3}$$

where the convention that $0 \log 0 = 0$, $p(x) \log \frac{p(x)}{0} = \infty$ if $p(x)>0$, and $0 \log \frac{0}{0} = \infty$ is used. The relative entropy $KL(p|q)$ is a measure of the inefficiency of assuming that the distribution is q when the true distribution is p [Cov91].

Information-Theoretic Viewpoint Selection Measures

Information-theoretic-based viewpoint selection metrics have been successfully applied in computer graphic areas, such as scene understanding and virtual exploration [Vaz01, Vaz03, Sbe05] and volume visualization [Bor05, Tak05]. In this section, we review the *viewpoint entropy* [Vaz01, Vaz03] and the *viewpoint Kullback-Leibler distance* [Sbe05] which have been used to compute the best viewpoints of a scene. Recently, the *viewpoint mutual information* has been introduced to select the

best views in volume rendering [Vio06] and for polygonal meshes [Fei06].

Viewpoint entropy, based on the Shannon entropy (1), has been defined [Vaz01] from the relative area of the projected polygons over the sphere of directions centered at viewpoint *v*. Thus, the viewpoint entropy was defined by

$$H_v = -\sum_{i=0}^{N_f} \frac{a_i}{a_t} \log \frac{a_i}{a_t}, \quad (4)$$

where N_f is the number of polygons in the scene, a_i is the projected area of polygon *i* over the sphere, a_0 represents the projected area of background in open scenes, and $a_t = \sum_{i=0}^{N_f} a_i$ is the total area of the sphere.

The best viewpoint is defined as the one that has maximum entropy.

In [Sbe05], a viewpoint quality measure based on the *Kullback-Leibler distance* (3) has been defined by

$$KL_v = -\sum_{i=1}^{N_f} \frac{a_i}{a_t} \log \frac{\frac{a_i}{a_t}}{\frac{A_i}{A_t}}, \quad (5)$$

where a_i is the projected area of polygon *i*, $a_t = \sum_{i=1}^{N_f} a_i$, A_i is the actual area of polygon *i* and $A_T = \sum_{i=1}^{N_f} A_i$ is the total area of the scene or object. In this case, the background is not taken into account. The minimum value 0 is obtained when the normalized distribution of projected areas is equal to the normalized distribution of actual areas. Thus, selecting high quality views means minimizing KL_v.

3. SIMPLIFICATION ERROR METRIC BASED ON VIEWPOINT ENTROPY

In this section, we present a new error metric based on the viewpoint entropy that can be used to evaluate the cost of an edge collapse and hence to drive the simplification process.

Viewpoint entropy (4) is based on the distribution of polygon areas seen from a viewpoint. Thus, if a polygon is not seen from any point of view its contribution to the formula is zero and the geometry that is hidden will initially be removed.

Given a particular viewpoint, we can consider the following: if a simplification is produced near the silhouette and increases the whole area seen, the overall value of viewpoint entropy will have been changed. So if we want to keep the silhouette of the model we must try to reduce this change. Note that the area of the background is included in the formula as polygon 0. This allows viewpoint entropy to preserve the silhouette better.

Due to the above seen characteristics of viewpoint entropy and the fact that it expresses the accessible information about the object from a given viewpoint, the variation of this measure for each viewpoint can provide us with an error metric to guide the simplification process. Taking into account these facts, the *simplification error metric* is defined by the sum of variations of viewpoint entropy for all viewpoints *V*:

$$c = \sum_{v \in V} \left| H_v - H'_v \right|, \quad (6)$$

where H_v is the viewpoint entropy before an edge collapse and H'_v is the viewpoint entropy afterwards. With respect to its computation, several techniques have been analyzed in [Cas06]. More specifically, the OpenGL histogram, the hybrid SW-HW histogram and the occlusion query were studied. The best technique was found to be the hybrid SW-HW histogram in current hardware. This technique takes advantage of the PCI Express bus symmetry. A different color is assigned to each polygon and the whole object is sent for rendering. Next, a buffer read operation is performed, and then this buffer is analyzed pixel by pixel to retrieve data about its color. Using an RGBα color codification with a byte value for each channel, up to 256^4 polygons can be calculated with only one single rendering pass. We used this technique during the simplification process.

4. SIMPLIFICATION ALGORITHM

The simplification process, like many other simplification algorithms, is based on the edge collapse operation. However, we use the half-edge collapse operation. According to this, the remaining vertex for an edge collapse *e(u,v)* is vertex *u* or *v* (Figure 1). By using half-edge collapses it is possible to reuse the simplification process in order to generate multiresolution models. These models can use the current hardware in a more efficient way because no new vertices are added to the original model. The main disadvantage is a slight loss of quality of the final mesh, although the complexity of the simplification algorithm is reduced because we do not have to compute the position of the new vertex *v'* resulting from the edge collapse.

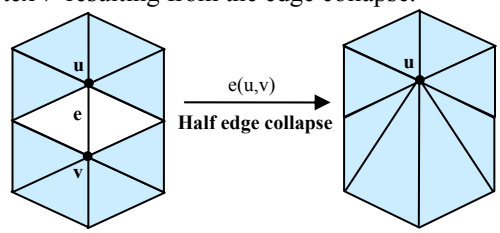


Figure 1. The half-edge collapse operation. In this example the edge *e(u, v)* is collapsed into vertex *u*, but could also be collapsed into *v*

We only take into account the edges that have at most two adjacent polygons, that is to say 2-manifold edges. And we also consider boundary edges, i.e. edges that have one single adjacent polygon.

 ISBN 978-80-86943-98-5

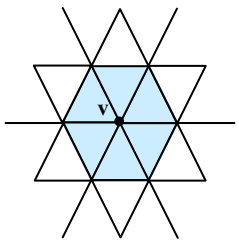


Figure 2. Edges adjacent to vertices adjacent to vertex *v*

This allows us to keep the shape of the mesh better during the simplification process. In addition, we compute the best half-edge collapse. To do this, we use the approach developed by Melax [Mel98], which takes into account polygon normals. We calculate the two possibilities *e(u,v)* and *e(v,u)* and finally we apply the direction that produces a minor change in the curvature of the local region around the edge collapse.

Edge Collapse Error

In the previous section, we defined the error induced by an edge collapse as the sum of the differences in viewpoint entropy before simplifying and after simplifying.

To speed up its calculation, we can make use of the fact that viewpoint entropy can be iteratively calculated. Viewpoint entropy is a calculation from the projected areas, and only a few polygons change after an edge collapse. So viewpoint entropy can be computed at the beginning for the entire object and then this initial viewpoint entropy can be successively updated. In our implementation we have exploited this feature.

We choose the edge collapse that has the least deviation *c* (6). It is important to determine some parameters, since the quality of the results could change. We have performed measurements with 20 regularly distributed viewpoints and rendered 256x256 resolution images. Higher values increase quality, but also significantly raise the temporal cost.

In [Cas05] different hardware techniques were analyzed for geometric visualization using standard OpenGL running on current GPUs. In this study it is shown that the vertex buffer objects technique is the best suited to dynamic geometry. So we used this technique to render our images.

We found that more accurate results are obtained with many viewpoints than with just a few, although the computational cost is obviously higher. At each iteration the edge cost has to be evaluated for the entire set of remaining edges. An edge collapse in our algorithm could, in principle, affect the cost of any remaining edge. But this does not always happen to each edge. At each iteration we only choose a small group of edges that are affected by an edge collapse and then the cost for these edges is recalculated. These edges are the ones that are adjacent to the vertices adjacent to the vertex *v* resulting from a half-edge collapse (Figure 2). In our experiments, if we consider the whole set of edges of the model, the temporal cost is increased around 20 times, but we obtain results that are not significantly better. In Figure 3 we show a summary of this algorithm.

```
/* Compute Viewpoint Entropy for the
original mesh M */
Compute H_v where v={1,..,n}
/* Build initial priority queue of edge
collapses */
for( e ∈ M )
  Perform collapse e
  Compute H'_v where v={1,..,n}
  Compute collapse cost c= Σ_{v=1}^{n} |H_v − H'_v|
  Insert the duple (e, c) in queue q
  Undo collapse e
end for
/* Update the mesh */
while (queue q not empty)
  Delete from queue q the edge e with lowest
  c
  Perform collapse of e
  Recalculate cost of every edge in the
  neighborhood of the transformation of e
  and update their location in queue q
end while
```

Figure 3. Pseudo-code of our viewpoint entropy-driven simplification algorithm

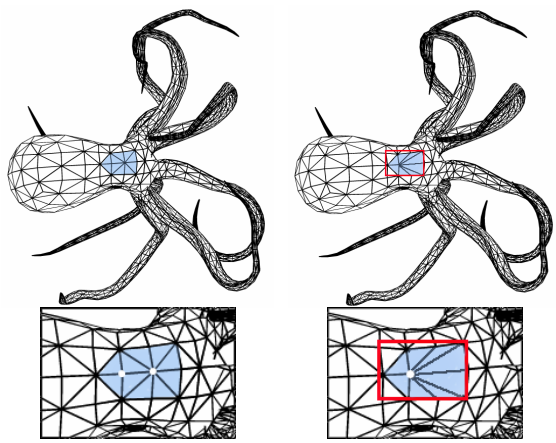
Updating Projected Areas

In order to compute the entropy of a viewpoint we need the projected areas of every polygon from this viewpoint. The bottleneck resides in the pixel-to-pixel analysis performed by the hybrid SW-HW histogram [Cas06] to obtain these areas, due to the memory transfer cost. Therefore, we can reduce this overload if instead of analyzing the whole image we restrict the area of reading to a window that only includes the polygons surrounding the edge collapse (Figure 4).

To obtain this window, first we determine the bounding box that includes the polygons surrounding the edge collapse and then we project this bounding box onto the screen.

This method allows us to reduce the temporal cost of the algorithm by around 10 times, but it can lead to some slight loss of quality. This is mainly due to the fact that after an edge collapse some hidden polygons may appear and we could not measure their contribution to the formula.

 ISBN 978-80-86943-98-5



(a) (b)

Figure 4. Image (a) shows the Octopus model, the triangles surrounding the edge collapse are marked in blue. Image (b) shows in red the window which is used to obtain the new projected areas for blue triangles

5. RESULTS

We carried out our tests with low complexity models from CAD programs. All models were simplified from 20 viewpoints on a Pentium Xeon 2GHz with 1GB RAM and an NVIDIA 7800 GTX 512MB graphics card. We compared the results obtained at the same simplification level to the results with QSlim [Gar97], a well-known geometric-based method and freely available, using the best half-edge collapse. The images shown were obtained using different viewpoints from those used during the simplification process.

Model	Triangles		Error	
	Original	Final	H_v	QSlim
Fish	815	100	0,05	0,09
Galleon	4 698	500	0,11	0,22
Fracttree	4 806	1 200	0,08	0,12
Galo	6 592	500	0,03	0,05
Octopus	8 468	500	0,09	0,16
Big_porsche	10 474	1 000	0,04	0,10
Unicycle	13 810	1 000	0,03	0,07

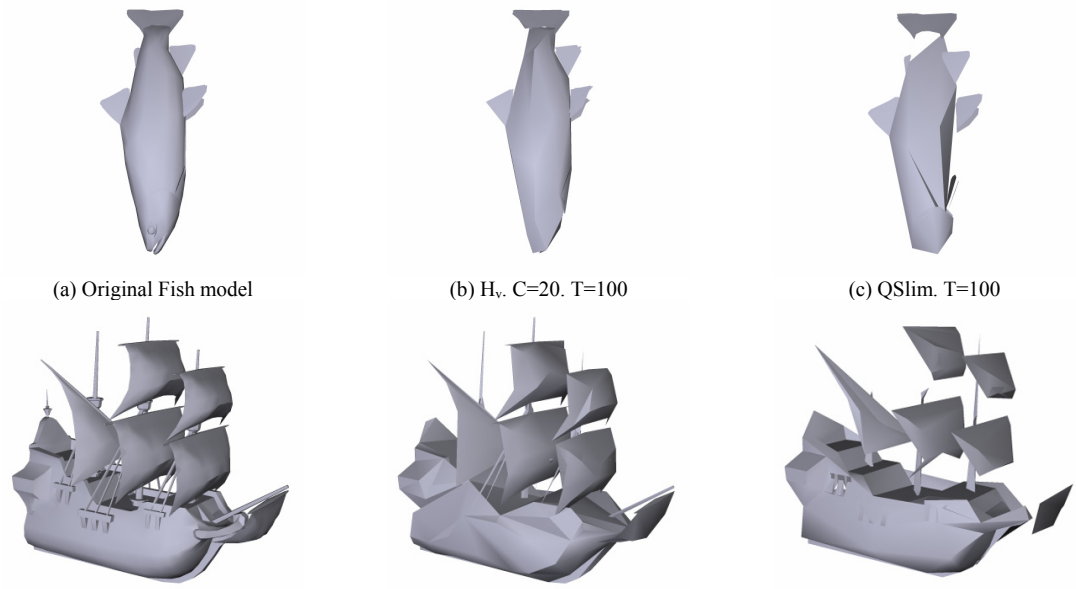
Table 1. Errors measured with Metro for all models

Model	Triangles		Error	
	Original	Final	H_v	QSlim
Fish	815	100	11,40	22,83
Galleon	4 698	500	17,74	36,84
Fracttree	4 806	1 200	30,19	34,10
Galo	6 592	500	9,03	12,40
Octopus	8 468	500	17,35	25,84
Big_porsche	10 474	1 000	7,48	8,28
Unicycle	13 810	1 000	10,32	11,06

Table 2. Errors measured with RMSE for all models

Model	Triangles		Time	
	Original	Final	H_v	QSlim
Fish	815	100	11.16	0.02
Galleon	4 698	500	92.64	0.06
Fracttree	4 806	1 200	96.30	0.08
Galo	6 592	500	152.29	0.08
Octopus	8 468	500	224.89	0.09
Big_porsche	10 474	1 000	299.47	0.13
Unicycle	13 810	1 000	451.76	0.20

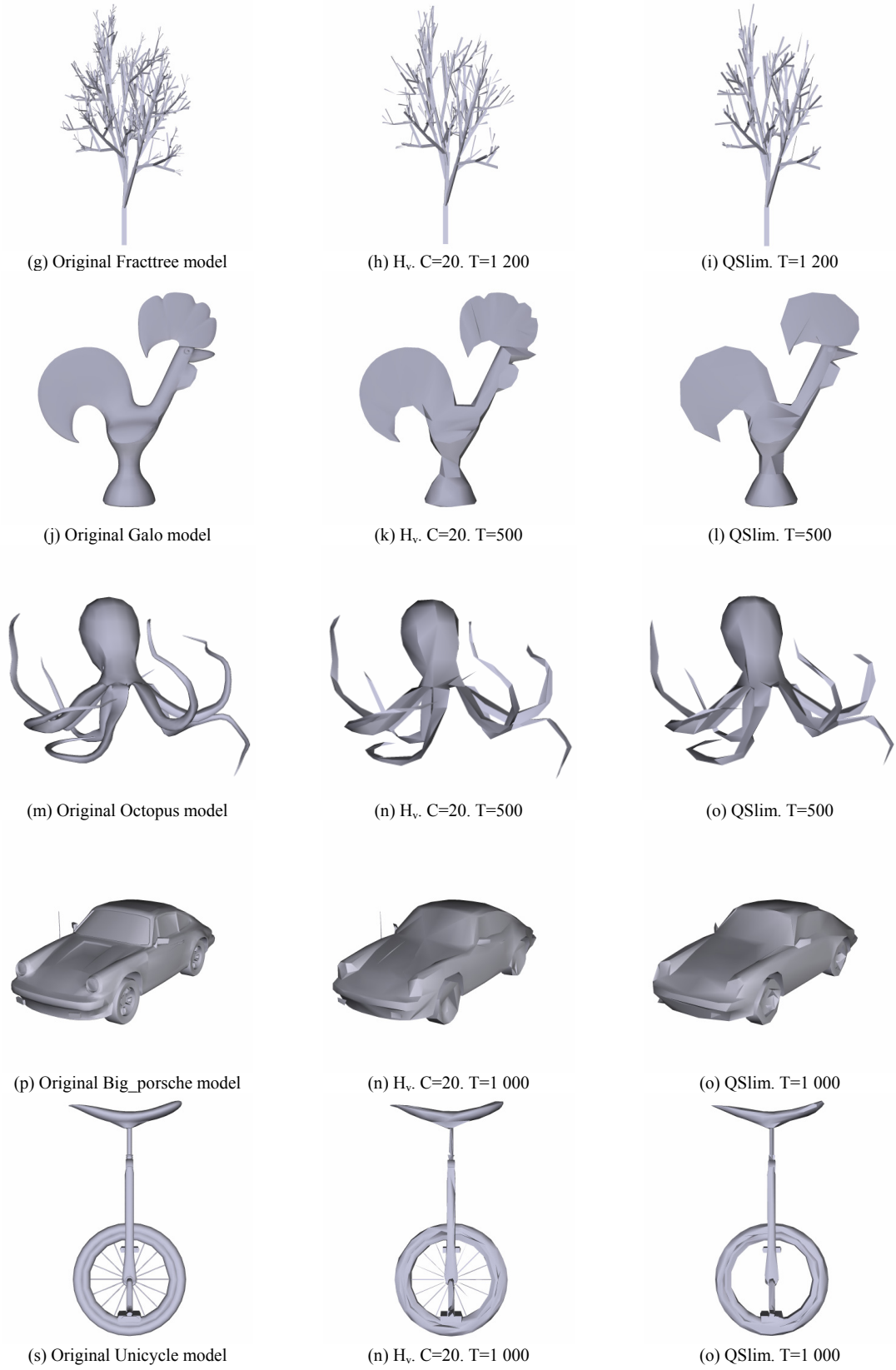
Table 3. Simplification times measured in seconds for all models



(a) Original Fish model (b) H_v. C=20. T=100 (c) QSlim. T=100

(d) Original Galleon model (e) H_v. C=20. T=500 (f) QSlim. T=500

 ISBN 978-80-86943-98-5



(g) Original Fracttree model (h) H_v. C=20. T=1 200 (i) QSlim. T=1 200

(j) Original Galo model (k) H_v. C=20. T=500 (l) QSlim. T=500

(m) Original Octopus model (n) H_v. C=20. T=500 (o) QSlim. T=500

(p) Original Big_porsche model (n) H_v. C=20. T=1 000 (o) QSlim. T=1 000

(s) Original Unicycle model (n) H_v. C=20. T=1 000 (o) QSlim. T=1 000

Figure 5. Results for all models. C indicates the number of viewpoints and T the number of triangles

In Table 1 and 2 we present the error committed in our experiments. Table 1 analyzes the visual error and Table 2 shows the geometric error. We have implemented the root mean square error (RMSE) of the pixel-to-pixel image difference defined in [Lin00] to measure the mean visual error between the

original and the simplified model. This error was taken using 24 viewpoints and 512x512 resolution images. We must emphasize that each viewpoint was different from the one used during the simplification and the resolution was higher. Clearly, the visual error committed in our method is quite low compared to QSlim, and can even be 50% lower, as shown in the case of the Fish model. We have also measured the geometric error using the mesh comparison tool called Metro [Cig98], and our results are rather better than the geometric method used for comparison purposes. This makes us highly confident about our approach. For example, the geometric error committed in the Galleon, Big_porsche and Unicycle models using H_v are 50% less than with QSlim.

In Table 3 we show an analysis of the temporal cost of our method. This cost is proportional to the complexity of the model and the final number of triangles demanded. However, the QSlim algorithm is extremely fast. Its times for these models are less than a second.

In Figure 5 we show the results for all the models analyzed. The H_v achieves much better simplification than QSlim. For example, in the Fish model the tail and the mouth shape is kept better, and in the Galleon model the same can be said for the sails and the masts. In the Fracttree model there are more branches, while in the Galo model the crest and the tail, in the Octopus model the tentacles, in the Big_porsche model the headlights and the aerial and, finally, in the Unicycle model the spokes are all far better represented.

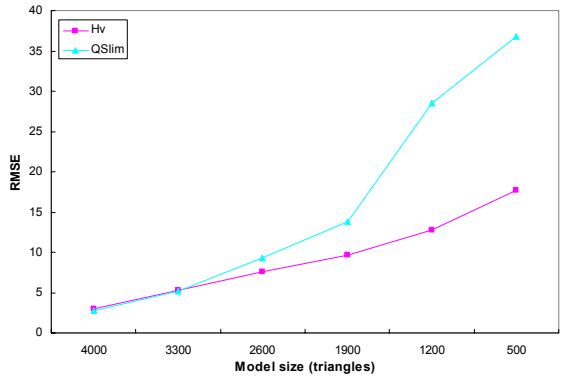


Figure 6. RMSE errors for the Galleon model

In Figures 6 and 7 we show how H_v acts at several degrees of simplification for the Galleon model. We have measured the RMSE and the geometric error. In Figure 6 we show that as we increase the level of simplification the difference between H_v and QSlim becomes larger and the visual quality of H_v is much higher. In Figure 7 we show that the geometric error of H_v is also lower than QSlim, except during the very first stages. This can be accounted for by the fact that H_v is a global measure, and it is possible that, in these stages, QSlim could often be better because it evaluates the error locally.

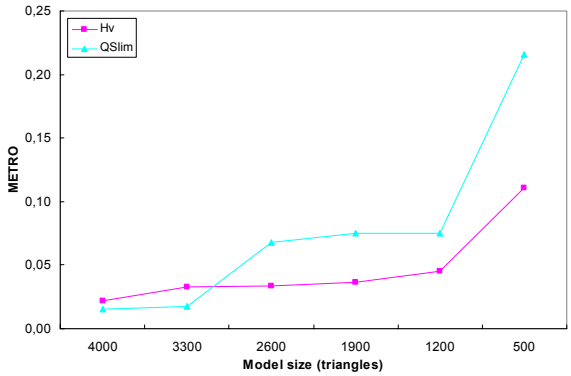
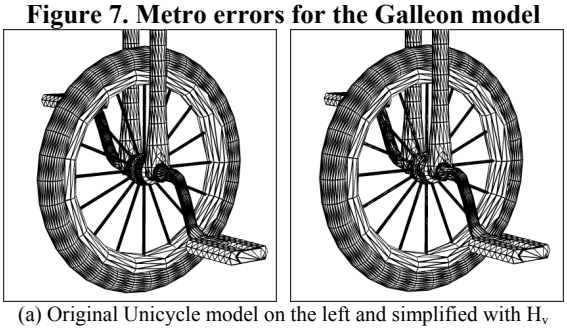
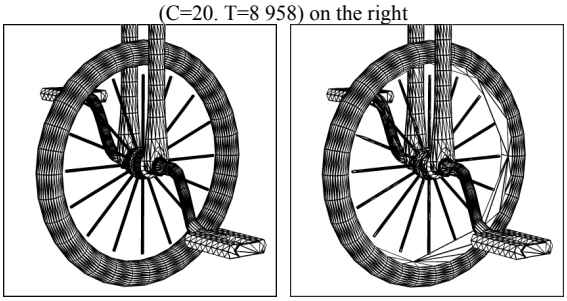


Figure 7. Metro errors for the Galleon model



(a) Original Unicycle model on the left and simplified with H_v (C=20. T=8 958) on the right



(b) Original Unicycle model on the left and simplified with H_v (C=20. T=8 958) on the right, both without rim

Figure 8. Close-ups of Unicycle model simplified with H_v at very early degree of simplification

Finally, in Figure 8 we show how the H_v acts at very early simplification levels for the Unicycle model. In this case we analyze the Unicycle model since it presents hidden interiors on the inner part of the tire which is in contact with the rim. As shown in this figure, H_v achieves a great level of simplification in this region (see 10(b) on the right). The model is simplified by around 35% and is visually the same. At this level, most simplifications focused on hidden interiors.

We also have conducted some experiments with more viewpoints and our results slightly improve but the temporal cost increases substantially. Therefore, we think that with 20 viewpoints we already have excellent results, and thus it is a good compromise between quality and efficiency.

 ISBN 978-80-86943-98-5

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a new mesh simplification method based on viewpoint entropy. Our method performs a simplification with lower visual and geometric error than QSlim. In addition, it achieves very good results with CAD models. These models are composed of different pieces that are assembled together, thus presenting a lot of hidden zones and this is where our algorithm hits harder. In general, the main drawback of image-based methods is the high temporal cost. Our approach, based on viewpoints, also has a high cost compared to geometric-based simplifications. However, we have shown that our method achieves better simplifications by taking into account visual similarity and even it improves geometric error in most cases.

Finally, it could be very interesting to make a study with other metrics based on information theory such as Kullback-Leibler distance and mutual information, which can also be applied to the simplification framework. And it could also be useful to incorporate mesh properties such as color and texture.

7. REFERENCES

[Bor05] Bordoloi U., Shen H.W. Viewpoint evaluation for volume rendering. Proc. of IEEE Visualization 2005, Minneapolis, USA, May 2005.

[Coh98] Cohen J., Olano M., Manocha D. Appearance preserving simplification. Proc. SIGGRAPH '98 (1998), vol. 32, pp. 115-122.

[Cas05] Castelló P., Ramos J.F., Chover M. A comparative study of acceleration techniques for geometric visualization. Proc. of ICCS 2005, pp. 240-247.

[Cas06] Castelló P., Sbert M., Chover M., Feixas M. Techniques for computing viewpoint entropy of a 3D scene. Proc. of ICCS 2006, pp. 263-270.

[Cig98] Cignoni P., Rocchini C., Scopigno R. Metro: measuring error on simplified surfaces. Computer Graphics Forum, vol. 17(2), pp. 267-274, June 1998.

[Cov91] Cover T.M., Thomas J.A. Elements of Information Theory. Wiley Series in Telecommunications, 1991.

[Fei99] Feixas M., del Acebo E., Bekaert P., Sbert M. An information theory framework for the analysis of scene complexity. Computer Graphics Forum (Eurographics'99 Proceedings, Milan, Italy), 18(3), pp. 95-106, 1999.

[Fei06] Feixas M., Sbert M., González F. A unified information-theoretic framework for viewpoint selection and mesh saliency. IIiA-06-06-RR, Institut d'Informàtica i Aplicacions, Universitat de Girona (2006).

[Gar97] Garland M., Heckbert P. Simplification using quadric error metrics. Proc. SIGGRAPH '97 (1997), vol. 31, pp. 209-216.

[Gar98] Garland M., Heckbert P. S. Simplifying surfaces with color and texture using quadric error metrics. IEEE Visualization 98, Los Alamitos, CA, USA, 1998, pp. 263-269.

[Hop96] Hoppe H. Progressive meshes. ACM SIGGRAPH 1996. New York, USA, pp. 99-108.

[Hop99] Hoppe H. New quadric metric for simplifying meshes with appearance attributes. IEEE Visualization 1999, October 1999, pp. 59-66.

[Lue01] Luebke D.P., Hallen B. Perceptually-driven simplification for interactive rendering. Proc. of the 12th Eurographics Workshop on Rendering Techniques (London, UK, 2001, pp. 223-234.

[Lin00] Lindstrom P., Turk G. Image-driven simplification. ACM TOG 19, 3 (July 2000), pp. 204-241.

[Lee05] Lee C.H., Varshney A., Jacobs D.W. Mesh saliency. ACM Trans. Graph. 24, 3 (2005), pp. 659-666.

[Mel98] Melax S. A simple, fast, and effective polygon reduction algorithm. Game Developer (Nov. 1998), pp. 44-48.

[Rig05] Rigau J., Feixas M., Sbert M. Shape complexity based on mutual information. IEEE International Conference on Shape Modeling and Applications (2005).

[Sbe05] Sbert M., Plemenos D., Feixas M., Gonzalez F. Viewpoint quality: Measures and applications. Computational Aesthetics in Graphics, Visualization and Imaging (2005).

[Tak05] Takahashi S., Fujishiro I., Takeshima Y., Nishita T. A feature-driven approach to locating optimal viewpoints for volume visualization. Proc. of IEEE Visualization 2005, (at Minneapolis, USA) (2005), pp. 495-502.

[Vio06] Viola I., Feixas M., Sbert M., Gröller E. Importance-driven focus of attention. IEEE Trans. on Visualization and Computer Graphics, Vol. 12, No. 5, September/October 2006

[Vaz01] Vázquez P.P., Feixas M., Sbert M., Heidrich W. Viewpoint selection using viewpoint entropy. Proc. of Vision, Modeling, and Visualization 2001 (Stuttgart, Germany) (2001), pp. 273-280.

[Vaz03] Vázquez P.P., Feixas M., Sbert M., Heidrich W. Automatic view selection using viewpoint entropy and its application to image-based modeling. Computer Graphics Forum (Dec. 2003).

[Wil03] Williams N., Luebke D., Cohen J.D., Kelley M., Schubert B. Perceptually guided simplification of lit, textured meshes. Proceedings of the 2003 ACM SIGGRAPH, New York, USA, pp. 113-121.

[Zha02] Zhang E., Turk G. Visibility-guided simplification. Proc. of IEEE Visualization 2002 (Nov. 2002), vol.31, pp. 267-274.

Dimensional Induced Clustering for Surface Recognition

Massimiliano B. Porcu
Dip.to Matematica e Informatica
Università di Cagliari
Via Ospedale, 72
I-09124, Cagliari, Italy
massi@dsf.unica.it

Riccardo Scateni
Dip.to Matematica e Informatica
Università di Cagliari
Via Ospedale, 72
I-09124, Cagliari, Italy
riccardo@unica.it

Abstract

Understanding when a cloud of points in three-dimensional space can be, semantically, interpreted as a surface, and then being able to describe the surface, is an interesting problem in itself and an important task to tackle in several application fields. Finding a possible solution to the problem implies to answer to many typical questions about surface acquisition and mesh reconstruction: how one can build a metric telling whether a point in space belongs to the surface? Given data from 3D scanning devices, how can we tell apart (and eventually discard) points representing noise from signal? Can the reached insight be used to align point clouds coming from different acquisitions?

Inside this framework, the present paper investigates the features of a new *dimensional clustering algorithm*. Unless standard clustering methods, the peculiarity of this algorithm is, using the *local fractal dimension*, to select subsets of lower dimensionality inside the global of dimension N.

When applied to the study of discrete surfaces embedded in three dimensional space, the algorithm results to be robust and able to discriminate the surface as a subset of fractal dimension two, differentiating it from the background, even in the presence of an intense noise. The preliminary tests we performed, on points clouds generated from known surfaces, show that the recognition error is lower than 3 percent and does not affect the visual quality of the final result.

Keywords: Surface representations; Point-based representation; Geometric algorithms.

1 INTRODUCTION

In the last few years, the diffusion of 3D data acquisition devices and scanning systems, boosted the use of huge volumes of point samples to model and represent real world objects [18]. At the same time, a big effort has been done, from the computational point of view, to improve algorithms and analytic techniques for point clouds processing, surface reconstruction, and semantic interpretation [11, 17].

In this context, the contribution of this paper regards the relations between a surface and the points cloud defining it. Particularly, we investigate on the possibility that there is some intrinsic feature of a points cloud, allowing us to discriminate a dataset defining a surface from the acquisition noise or any other uncorrelated point subset. We define *intrinsic feature* something that is independent from any particular reconstruction we can arrange, and can be evaluated directly on the points set. Some important topics are strictly related to this problem: given a points cloud defining a surface, can we figure out whether a new point belongs to the same set of the others or not (is a point on the surface or not)? Can we select, and discard, mismatched points (error outliers)? How many points can we remove from a surface, without damaging its intrinsic existence (decimation)?

As a research starting point, we can observe that the essential peculiarity of a surface is to be a subset of intrinsic dimension two. Several clustering methodologies has been proposed lately for the detection, inside a higher-dimensional dataset, of subsets of points whit lower intrinsic dimensionality [10, 4, 16]. These techniques are then capable to discover, into the global set, points arranged along lines (1D subsets), over surfaces (2D subsets) or hypersurfaces (nD subsets).

A recently proposed algorithm, the *Dimensional Induced Clustering* algorithm (DIC), appears to be very effective at doing this [10]. Based upon the evaluation of the intrinsic fractal dimension and the local point density, it is able to discriminate between subsets of different dimensions, and also of the same dimension but different density.

 ISBN 978-80-86943-98-5

In this paper, we investigate on the use of DIC for the discrimination of points clouds in three dimensional space, and on its consequences. The rest of this work is organized as follows: in Section 2 we briefly go over the previous work done in points clouds processing and clustering techniques; in Section 3 we illustrate the Discrete Induced Clustering Method; then, we describe the use of DIC for the study of surface points clouds, introducing the general investigation framework and going into details in Section 4; in Section 5 we show the results obtained using the algorithm on several meshes widely used in literature; finally in Section 6 we draw our conclusions and describe the future evolutions of this work.

2 PREVIOUS WORK

Several techniques have been proposed for the reconstruction of surfaces from oriented point samples.

The problem of reconstructing a surface obtained from 3D scans is mainly reduced to the registration of the set of regular maps (the points are placed on the vertices of a regular grid) since the relative positions of the single maps is known by the acquisition planning [9]; under normal conditions there is low digital noise and it is interesting to understand where the map superpositions are.

The most diffused approaches when dealing with a set of *really*[1] unorganized points are based on topology. They use alpha shapes [12] or three-dimensional Voronoi diagrams of the point set, and then use the facets of the tessellation for the reconstruction of the surface [2]. The latter technique was recently reformulated to be also able to filter out small perturbations (noise) near to the target surface [7].

Density-based clustering methods uses local density information in order to partition a dataset (it could be a point set). In this context, Hierarchical Single Linkage is a widely diffused technique, implemented in common packages as DBSCAN [8], OPTICS [3] and CLIQUE [1].

The intrinsic fractal dimension has been successfully used in numerous database selection problems, as the nearest-neighbor queries [19] and spatial query selective estimation [5]. Recently, also some partitioning and clustering methods based upon this technique have been proposed [4].

3 THE DIC ALGORITHM

Clustering methods search for patterns and regularities in data, to perform partitions in homogeneous sets. The Dimension Induced Clustering algorithm (DIC) [10] carries this out computing the intrinsic dimension and the point density. It is therefore designed to detect, inside a higher-dimensional dataset, subsets of points whit lower intrinsic dimensionality, such as lines (1D subsets) or surfaces (2D subsets). Moreover, the use of local point density information makes the method being able to discriminate between subsets of the same intrinsic dimension.

3.1 Fractal Dimension

Behind the word *dimension* of a set there are actually two different meanings that may not be coincident. The first one is the number of independent coordinates used to locate, in unambiguous way, a point of the set. The second one is the relative growing rate of the point set, that is to say, the *speed* the point set fill the space with. This is in fact the *fractal dimension*. This concept can be explained using the drawing of Figure 1. Consider the seed A and count the number of points of the set inside the ball of radius R: doubling the ball radius, we expect to find twice the number of points, so the fill rate grows linearly with the radius and thus the fractal dimension of the point set around A is one. If we consider, instead, the seed B, we expect to have a quadratic fill rate and thus the fractal dimension of the point set around B is two. We can also have a non-integer fractal dimension: using, for instance, as seed the point C we obtain a fractal dimension of 1.5.

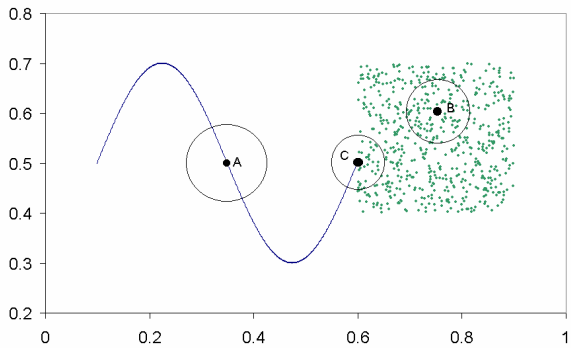


Figure 1: Fractal dimension interpreted as the relative growing rate of a point set.

Consider now the dataset $X \subset \mathbf{R}^m$ and assume that the number of points n of X tends to infinity. Let choose a distance function on X and let $B(x,r)$ be the ball of radius r centered at x; let $|B(x,r)|$ be the number of points inside the ball. We define the *local growth function* for the point x as:

$$G_x(r) = \lim_{n\to\infty} \frac{1}{n}|B(x,r)|$$

The local growth function represents the density of the subset $B \subset X$ for a fixed r; it contains in-

[1] We mean that there is no implicit information on the adjacencies as in 3D scanning.

 ISBN 978-80-86943-98-5

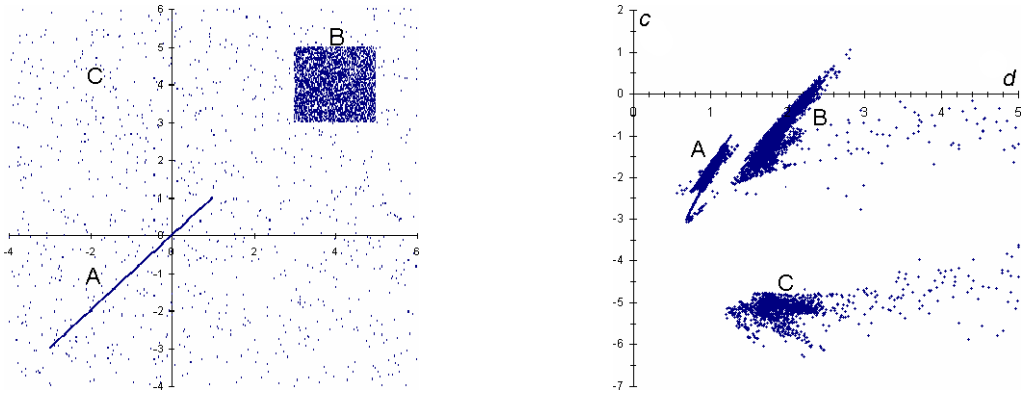


Figure 2: The DIC algorithm is able to discriminate between: (i) datasets of different dimensions; (ii) datasets of the same dimension but different densities.

formation about the growth rate of the number of neighbors of x. The local fractal dimension d_x of a point x is defined as:

$$d_x = \lim_{r,r' \to 0} \frac{\log \frac{G_x(r)}{G_x(r')}}{\log \frac{r}{r'}}$$

Since we deal with finite sets, to evaluate the fractal dimension d_x we define the local growth curve as the fraction of points inside the ball for a chosen radius, $G_x(r) = \frac{|B(x,r)|}{n}$, and compute d_x of a point x as the slope of the curve $G_x(r)$ in log-log scale. Practically we can do that assuming a linear model for the curve, and finding the line that fits it best, using a least squares method. To obtain a correct local fit we need to consider the function $G_x(r)$ in a region not too much distant from the center x (for details see [10]). Using the symbol L_x to denote the curve, our linear model produces a fitting curve of the form:

$$L_x(\log r) = d_x \log r + b_x$$

3.2 Local Density

For any value of x there is a particular value r^* for the radius leading to the definition of the coefficient c_x:

$$c_x = L_x(\log r_x^*) = d_x \log r_x^* + b_x$$

We can take c_x as a measure of the density of the dataset in a ball of radius r^* centered on x and thus the radius r^*, varying from point to point, is the unique value minimizing the correlation between d_x and c_x and therefore maximizing the information contained in the couple of coefficients. It is possible to prove that this condition is obtained choosing:

$$\log r_x^* = \frac{\sum (d_x - \overline{d})(b_x - \overline{b})}{\sum (d_x - \overline{d})}$$

where $\overline{d}$ and $\overline{b}$ are the arithmetic means of the coefficients d_x and b_x computed on the whole dataset.

3.3 Local Representation

Using this procedure, we can thus define a mapping $f : X \to \mathbf{R}^2$ from the m-dimensional dataset X to the plane: we call it the *Local Representation* ($\mathcal{LR}$) of X. This mapping projects each element of the set into the 2D plane using the couple of coefficients (d, c), that are, respectively, the fractal dimension and the local points' density. Once projected the starting dataset X to its $\mathcal{LR}$, the partitioning task becomes definitely easier: we can, in fact, analyze and partition the $\mathcal{LR}$ and then project back the obtained results to the original dataset. The partition on the $\mathcal{LR}$ can be automatic, applying some standard clustering algorithm like Expectation Maximization (EM) [13], or interactive by visual inspection.

3.4 DIC Trials

Using the co-operating descriptors (d, c), DIC appears to be very effective since it is able not only to discriminate between subsets of different dimension, but also to distinguish subset of the same dimension but different densities. An example is given in Figure 2. The starting 2D dataset (left) is a mixture of three different types of regions: a 1D line (region A) a 2D dense cloud (B) and diffuse ground noise (C). B and C have the same dimension while A and B have the same density.

As a result of the application of the DIC algorithm, the set is projected to its $\mathcal{LR}$ (right). Notice the presence of 3 well separated clouds of points: two of them show a fractal dimension $d \approx 2$ while the third shows $d \approx 1$. The back projection shows that the clouds correspond, respectively, to the subsets A, B and C in the original point set. Using only the fractal dimension information, it is not possible to tell B and C apart; instead, using only a measure for the density, subsets A and B are not separable.

 ISBN 978-80-86943-98-5

4 SURFACE RECOGNITION

Now we want to focus our attention on the main contribution of this paper: how to use the DIC algorithm for surface recognition and decimation.

One possible scenario is as follows: consider to have to deal with a 3D points dataset of possible unknown origin sampling a surface in space, that can also be affected by a high experimental error or be completely intermixed with ground noise points. You want to be able, first, to reconstruct the surface, then to capture its saliencies using possibly unsupervised methods. Another alternative scenario can instead be the alignment of several range maps coming from 3D scans. We will see how we can use DIC for both these purposes.

To experiment on the use of the DIC algorithm we first modified well known benchmark datasets adding artificial noise to them. We embedded the datasets in a space region in which is present white noise constituted by a set of points, without any special regularity, randomly distributed with homogeneous density. The noise was produced using a standard random number generator.

To represent a correlate set of points, as for example a surface, a subset of the dataset should have some intrinsic features allowing us to distinguish it from the ground noise. Particularly, using the DIC approach, we want:

- DIC to be able to separate the data set from the ground noise, using dimension and/or density;
- The local fractal dimension of the data set points be correct (for a surface, $d \approx 2$).

This point of view is analogous to a signal-noise ratio evaluation. The starting data set being the signal, that we suppose to be two-dimensional, and the added points being the background noise, three-dimensional by construction.

Let now step over the different tasks performed on the dataset.

4.1 Surface Finding

In Figure 3 we represent a point set embedded in a noisy 3D space region. We constructed the point set using the vertices of a triangle mesh commonly used as a benchmark mesh (Oilpump, on the left). Starting from a surface, we know for sure that the points cloud does define a surface. We then introduced random noise, adding a number of points equal to half the points on the surface, chosen at random inside a box twice the volume of the surface's bounding box (on the right).

The global set of points, (surface + noise), has been processed with DIC, producing its Local Representation $\mathcal{LR}$ (in Figure 4). The possibility to separate the signal from the noise, and the computational techniques used to do it, depends on the features of $\mathcal{LR}$. In the diagram we can notice the presence of two well separated clusters, and of some outliers. The cluster named N, with a lower value of c, is centered around $d \approx 3$; the cluster S, with greater c, is centered around $d \approx 2$. S and N have spheroidal shape, are well separated from each other and overlap just in the outlier region. Under these conditions, it is possible to subdivide the $\mathcal{LR}$ plot in clusters, project them back on the 3D points generating them, and then use the clusters to partition the original 3D dataset. This procedure can be completely unsupervised using a standard partition method. We adopted, in this case, the K-Medoids algorithm [14].

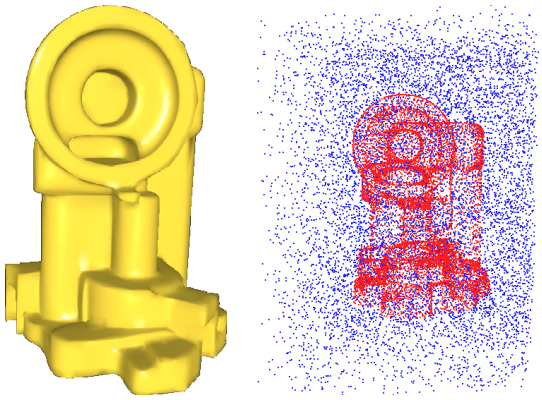


Figure 3: The Oilpump data set. The original surface on the left, the point set with white noise added on the right.

A backtrace check for clusterization procedure is possible, because the membership for the points, surface or noise, is known by construction. It shows that DIC discriminates correctly the noise cloud (N) from the surface point set (S). In fact, N has a mean fractal dimension $d \approx 3$ and lower density c and it is contains almost only noise points. S has dimension $d \approx 2$ and higher density and it is mainly constituted of surface points. In our experiments, as it will be explained in further detail in Section 5, uncorrect assignments are bounded by less than 3 percent for false negatives and by 6 percent for false positives.

4.2 Noise Raising

The surface finding procedure has been applied on datasets with different noise levels, changing the noise density. The back projections show a remarkable capability of DIC to discriminate the surface point set, also in challenging environmental conditions.

It's worth to remember that the signal stability, when the level of background noise raises, is

 ISBN 978-80-86943-98-5

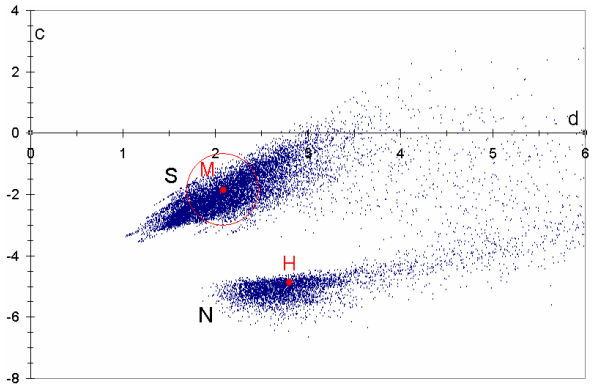


Figure 4: The Local Representation for the Oilpump dataset: in red the medoids M ($d = 2.156, c = -2.038$, signal) and H ($d = 2.764, c = -5.025$, noise).

a measure of the consistency and the reliability of the signal itself. Applied to our problem, it means that the likelihood that the point set defines a surface is directly correlated to the background noise level. We can evaluate this property for every single point of the data set. Embedding the data set in 3D white noise, some points will be mingled with the ground jelly, these points will be easily not classified as belonging to the surface by a reconstruction algorithm. On the contrary, the points that are classified as surface points even for high level of ground noise are the subset of strongly attached points, that is the intrinsic skeleton of the surface.

4.3 Points Decimation

In the last years, modelling and visualization techniques using points as primitives, emerged as interesting alternatives to traditional triangle mesh-based processing [22, 23, 15, 21]. In this frame it becomes important to set up algorithms capable to conveniently reduce the number of point samples while keeping the underlying shape features unchanged. Recently some methods has been proposed, based upon the intrinsic feature of the points cloud, mainly related to sample distribution and density [2, 6, 20].

We investigated the use of the DIC algorithm as a tool to perform a point selection based on intrinsic features like the fractal dimension d and the local density c.

To do this we adopted the following scheme: the clusterization method K-Medoids, used to separate the clouds on $\mathcal{LR}$ plot, works selecting a seed for each cluster, named *medoid*, that is representative of the cluster itself. The method assigns each element to the right medoid, minimizing the expectation value of the distance, with respect to some metric functions [13]. We can then select, for a cluster in the $\mathcal{LR}$ plot, a set of points closer to the relative medoid; for instance, inside a ball of some radius r.

Performing this procedure for the cluster representing the signal, the points we select by construction are:

- Representative of the surface;
- Homogeneous in density;
- Homogeneous in intrinsic dimension;
- Well separated from the noise.

We show an example for the Oilpump dataset in Figures 4 and 5. Using the information from the $\mathcal{LR}$ plot, we select a set of points of the signal by fitting a ball around the medoid M. The point density, as function of the distance from M (in Figure 5), shows that the most part of the surface, over 99%, is contained inside a ball of radius $r = 1.5$.

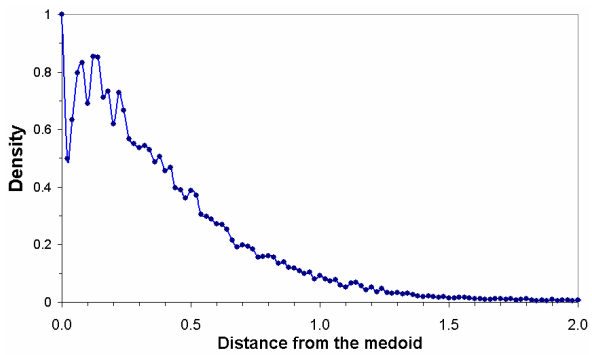


Figure 5: The normalized relative density of the points plot in Figure 4, as function of the distance from M.

In Figure 6 there are four examples of point subsets obtained using balls of different radius to select points from the dataset.

4.4 Superposition and Alignment

While reconstructing surfaces from 3D scans, different views need to be correctly aligned and overlapped in the common borders regions [9]. This task is complicated by the acquisition errors that are a further unknown of the problem.

Even if the type of error introduced by the acquisition process is not the random white noise that our method is best suited for, the DIC algorithm can still provide a support to existing techniques. Consider the really trivial example shown in Figure 7. Two planar surfaces orthogonal to z, with the same point density, are embedded in a 3D noisy region. Initially the surfaces are partially overlapped on xy, but distant in z (top left). Modifying the position in z of one plane, we generate a clouds overlap (top right). The $\mathcal{LR}$ plots relative to the two different cases are shown in the same

 ISBN 978-80-86943-98-5

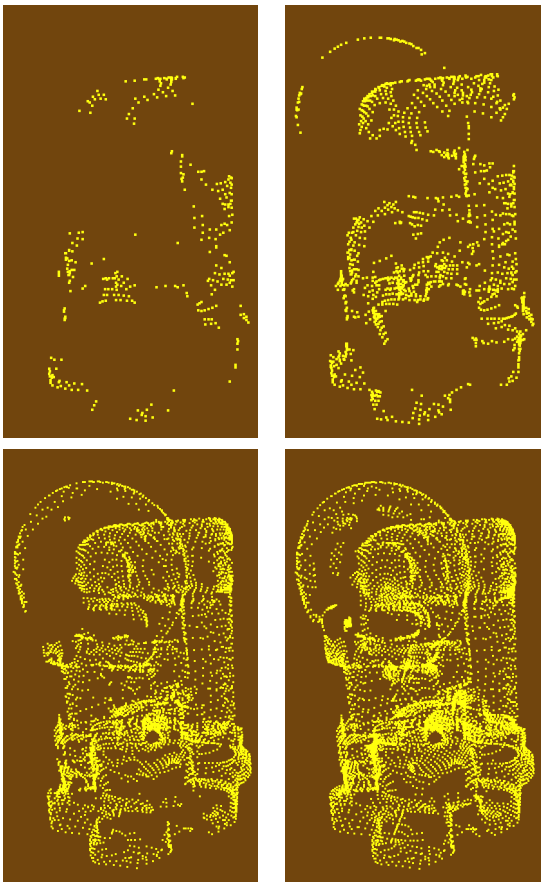


Figure 6: Oilpump points decimation, obtained using the medoid distance technique. From left to right and top to bottom, the subsets are generated using, respectively, $r = 0.3$, $r = 0.5$, $r = 0.8$, and $r = 1.1$.

picture (down left), where red means the overlap case and blue the non-overlap one. When there is no overlap the $\mathcal{LR}$ shows just two clusters since the planes have the same intrinsic dimension and the same density. In this case, DIC can discriminate them from the background noise but cannot create any link from one to another.

In the second case, instead, the points density in the overlapping region is twice the density of each plane. In this situation, DIC is capable to figure out if there is a superposition and also to figure out where the overlapping region is, as one could expect when they represent the same swatch of surface scanned twice. As one can see in the two diagrams, the $\mathcal{LR}$ plot for overlapping surfaces (red points) shows a third cluster, with the same fractal dimension of the blue signal cloud but higher density. As the analysis points out, this cluster is made of points in the superposition region. Being able to separate the signal from the noise, this mechanism correctly characterize the overlapping region taking into account just the points on the surfaces, and discarding the values affected by errors.

In this case the cluster separation is not simple using the K-Medoids method and user intervention is needed. It is, anyway, the only case we found in which the method needs to be supervised.

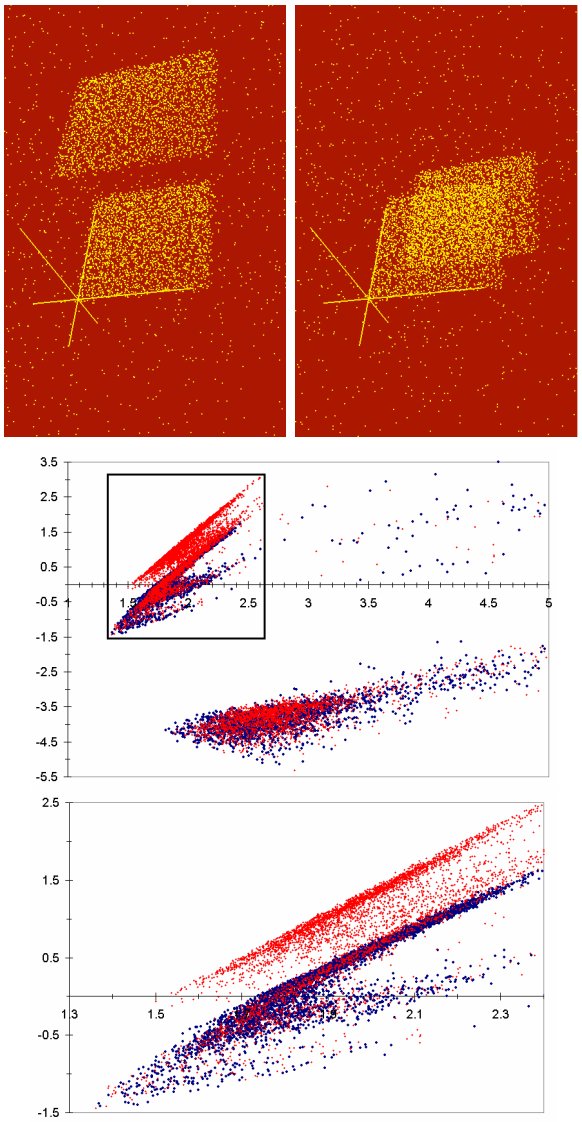


Figure 7: Two planes embedded in background noise. Not overlapped (top left), overlapped (top right). The related $\mathcal{LR}$ (in the middle, red for overlapped and blue for non overlapped) and a detail of the boxed region (bottom).

5 RESULTS AND DISCUSSION

In Table 1 we summarize the results of the discrimination between surface and noise, performed for different meshes. Noise was randomly generated adding points in an amount of 50% the number of points of the original mesh, inside a box eight times the volume of the mesh bounding box. We list, in the order, the name of the mesh, the number of its points and the number of added points. With

 ISBN 978-80-86943-98-5

Table 1: Noise/Surface discrimination for eight different datasets: in column V there is the number of points from the surface and in column N the number of added points.

Dataset	V	N	α	$\alpha_{\%}$	β	$\beta_{\%}$
Screw	904	452	14	1.5	55	6.0
Bunny	1015	507	2	0.2	155	1.5
Face	2278	1139	0		118	5.1
Ahead	3123	1561	89	2.8	124	3.9
Teapot	4255	2127	56	1.3	163	3.8
Cat	4539	2269	0		198	4.3
Fandisk	6475	3237	0		293	4.5
Oilpump	10274	5137	97	0.9	276	2.6

α we indicate the false negatives (surface points assigned to the noise set) and with β the false positives (noise points assigned to the surface). In columns $\alpha_{\%}$ and $\beta_{\%}$ we list the error percentages.

We can notice that the number of uncorrect assignments is small for both error types. It is very important that the number of false negatives is bounded by 3% and is, on average, much less. This implies that almost all the points to be used for the surface recognition are selected. The false positives are much more, because while dropping points at random in the bounding box the probability to choose spatial locations very close to the surface is not negligible.

In Table 2 we listed $\alpha_{\%}$ and $\beta_{\%}$ as function of the noise points density, for the Oilpump and Teapot datasets. The noise (first column) is expressed in a relative way: we start putting in a box, linearly twice the mesh bounding box, an amount of 50% the points of the mesh; then, we shrink the noise box to increase its density. Assuming 1 as a reference starting density value, the other values represent the relative growth. We can notice that $\alpha_{\%}$ is not related to the noise density; this means that the efficiency of the method is high also in challenging conditions. On the other hand, $\beta_{\%}$ grows more than linearly with the density. This is due to the fact that, reducing the noise box, the probability to generate random points located on, or very close to, the surface increases. In some sense then, a part of $\beta_{\%}$ is not really an error!

6 CONCLUSIONS AND FUTURE WORK

We presented in this work the application of a clusterization method to the recognition of surfaces from point clouds. The algorithm is based on the computation of the fractal dimension and local density for each point in the cloud (its $\mathcal{LR}$ map).

The main advantages of the method are: it can be used in an unsupervised mode using a standard clustering algorithm; it can be used interactively allowing for visual inspection of the characteristics of the dataset; based on the $\mathcal{LR}$ map a point decimation is direct and simple; a straightforward extension of the technique can be used for computing the superposition of different range maps coming from a 3D object scan.

Table 2: Variations of $\alpha_{\%}$ and $\beta_{\%}$ when changing the noise density.

Noise density	Oilpump		Teapot	
	$\alpha_{\%}$	$\beta_{\%}$	$\alpha_{\%}$	$\beta_{\%}$
1.00	0.96	1.22	1.39	1.57
1.18	0.98	1.47	1.27	2.11
1.42	0.98	1.94	1.29	2.84
1.72	0.94	2.69	1.32	3.83
2.12	0.94	3.51	1.36	4.61
2.65	0.91	5.53	1.18	6.56

The results, even if preliminary, are very promising, especially those regarding the point decimation aspect. We still need to work on several aspects of the method, the most important being: rework the implementation to be able to deal with larger datasets; make experiments on real sets of unorganized points instead of benchmarking on points clouds derived from meshes.

Nevertheless we are confident that it can already be useful for the purposes exposed.

REFERENCES

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.

[2] Nina Amenta and Marshall W. Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[3] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *SIGMOD Conference*, pages 49–60, 1999.

[4] Daniel Barbarà and Ping Chen. Using the fractal dimension to cluster datasets. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 260–264, New York, NY, USA, 2000. ACM Press.

[5] Alberto Belussi and Christos Faloutsos. Self-spacial join selectivity estimation using fractal concepts. *ACM Trans. Inf. Syst.*, 16(2):161–201, 1998.

 ISBN 978-80-86943-98-5

[6] Tamal K. Dey, Joachim Giesen, and James Hudson. Decimating samples for mesh simplification. In *CCCG*, pages 85–88, 2001.

[7] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. *Comput. Geom.*, 35(1-2):124–141, 2006.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[9] Thomas Franken, Matteo Dellepiane, Fabio Ganovelli, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Minimizing user intervention in registering 2D images to 3D models. *The Visual Computer*, 21(8-10):619–628, 2005.

[10] Aristides Gionis, Alexander Hinneburg, Spiros Papadimitriou, and Panayiotis Tsaparas. Dimension induced clustering. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 51–60, New York, NY, USA, 2005. ACM Press.

[11] Enrico Gobbetti and Fabio Marton. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28(6):815–826, December 2004.

[12] Baining Guo, Jai Menon, and B. Willette. Surface reconstruction using alpha shapes. *Comput. Graph. Forum*, 16(4):177–190, 1997.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer-Verlag, New York, 2001.

[14] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data. an introduction to cluster analysis.* Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics, New York: Wiley, 1990, 1990.

[15] Leif P. Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, December 2004.

[16] Robert Krauthgamer and James R. Lee. The black-box complexity of nearest neighbor search. In *ICALP*, pages 858–869, 2004.

[17] Jens Krüger, Jens Schneider, and Rüdiger Westermann. Compression and rendering of iso-surfaces and point sampled geometry. *The Visual Computer*, 22(8):517–530, 2006.

[18] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 131–144, July 2000.

[19] Bernd-Uwe Pagel, Flip Korn, and Christos Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *ICDE*, pages 589–598, 2000.

[20] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of IEEE Visualization 2002*, pages 163–170. IEEE Press, 2002.

[21] Mark Pauly, Leif P. Kobbelt, and Markus Gross. Point-based multiscale surface representation. *ACM Transactions on Graphics*, 25(2):177–193, April 2006.

[22] Miguel Sainz and Renato Pajarola. Point-based rendering techniques. *Computers & Graphics*, 28(6):869–879, December 2004.

[23] Ingo Wald and Hans-Peter Seidel. Interactive ray tracing of point-based models. In *Symposium on Point-Based Graphics 2005*, pages 9–16, June 2005.

 ISBN 978-80-86943-98-5

A Benchmarking Suite for Static Collision Detection Algorithms

Sven Trenkel
TU Clausthal, Germany
trenkel@tu-clausthal.de

Rene Weller
TU Clausthal, Germany
weller@in.tu-clausthal.de

Gabriel Zachmann
TU Clausthal, Germany
zach@in.tu-clausthal.de

ABSTRACT

In this paper, we present a benchmarking suite that allows a systematic comparison of pairwise static collision detection algorithms for rigid objects. The benchmark generates a number of positions and orientations for a predefined distance. We implemented the benchmarking procedure and compared a wide number of freely available collision detection algorithms.

Keywords: Collision Detection, Benchmarking

1 INTRODUCTION

Fast algorithms for collision detection between polygonal objects are needed in many fields of computer science, e.g. in physically based simulations, computer games, or robotics. In many of these applications, collision detection is the computational bottleneck. In order to gain a maximum speed of applications, it is essential to select the best suited algorithm.

There are a number of algorithms for collision detection between rigid objects. Unfortunately, it is extremely difficult to evaluate and compare collision detection algorithms, because in general they are very sensitive to specific scenarios, i.e. to the relative size of the two objects, the relative position to each other, the distance, etc.

The design of a standardized benchmarking suite for collision detection would make fair comparisons between algorithms much easier. Such a benchmark must be designed with care, so that it includes a broad spectrum of different and interesting contact scenarios. However, there are no standard benchmarks available to compare different algorithms. As a result, it is non-trivial to compare two algorithms and their implementations.

In this paper, we propose a simple benchmark procedure which eliminates these effects. It has been kept very simple so that other researchers can easily reproduce the results and compare their algorithms.

The user only has to specify a small number of parameters, namely: The objects he wants to test, the number of sample points, and, finally, a set of distances. Our algorithm then generates the required number of test positions and orientations by placing the object in the given distances.

Our benchmarking suite is flexible, robust, and it is easy to integrate other collision detection libraries. Moreover, the benchmarking suite is freely available and could be downloaded together with a set of objects in different resolutions that cover a wide range of possible scenarios for collision detection algorithms, and a set of precomputed test points for these objects [1].

2 RELATED WORK

There does not exist much work about special benchmarking suites for collision detection algorithms. Most authors simply choose some objects and test them in a not further described way, or they restrict their explorations just to some special scenarios. A first approach for a comprehensive and objective benchmarking suite was given by [Zac98]. The code for the benchmark is freely available. However, it does not guarantee to produce results with practical relevance, because the objects interpenetrate heavily during the benchmark, but collision detection is mostly used to avoid interpenetrations. In many simulations, objects are allowed to collide only a little bit, and then the collision handling resolves the collision by backtracking or a spring-damping approach.

[OL03] chose a set of physically based simulations to test their collision detection algorithms. This scenarios are a torus falling down a spiral peg, a spoon in a cup, and a soup of numbers in a bowl. [vdB97] positioned two models by placing the origin of each model randomly inside a cube. The probability of an intersection is tuned by changing the size of the cube. The problem here is that it is stochastic, and that a lot of large and irrelevant distances are tested.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency - Science Press, Plzen, Czech Republic.

[1] http://cg.in.tu-clausthal.de/research/colldet_benchmark/

 ISBN 978-80-86943-98-5

[CRM02] presented a comparison with the special focus on motion planing. They used different scenes in their probabilistic motion planner for the benchmark. However, such a benchmarking suite compares the collision detection algorithms only in a special scenario. In this paper, we present a more flexible and general benchmarking suite which produces more reliable comparisons.

3 THE BENCHMARKING ALGORITHM

Nearly all collision detection libraries for static collision detection between rigid objects are based on bounding volume hierarchies (BVHs). If the bounding volume (BV) of an object does not intersect a volume higher in the tree, then it cannot intersect any object below that node. So, they are all rejected very quickly. If two objects overlap, the recursive traversal during the collision check should quickly converge towards the colliding polygon pair. So, it is most time consuming if the BVHs overlap, but the objects do not.

Therefore, in most collision detection algorithms, the testing time depends mainly on the configuration of the two objects and their shapes, i.e. the positions, orientations and distance, and to a lesser amount on their complexity. Therefore, it seems to be reasonable for a well-balanced benchmarking procedure to test as many configurations for a given distance as possible.

3.1 The Search Space

Without loss of generality, it is sufficient to rotate only one of the objects in order to get all possible configurations, because we can simply transform one of the objects into the coordinate system of the other. This does not change the relative position of the objects. Therefore, our search space has 6 dimensions.

As even a 6D search space is too big to be tested, we have to reduce it by sampling. In order to find a large number of sampling points, we propose two different methods in our benchmarking suite. We call them *sphere method* and *grid method*. The sphere method is faster, but could miss some interesting configurations, while the grid method is more accurate. Both methods start with a fixed rotation. After a cycle of method-specific translations, the moving object is rotated and the next cycle can start until a user specified number of rotations is reached.

The Grid Method The first method uses a simple axis-aligned grid to find the translations. The center of the moving object is moved to the center of all cells. For each of these, the object is moved towards the fixed object until the required distance is reached. Then, the configuration is stored. Unfortunately, it is not possible to know the number of configurations found by this method in advance.

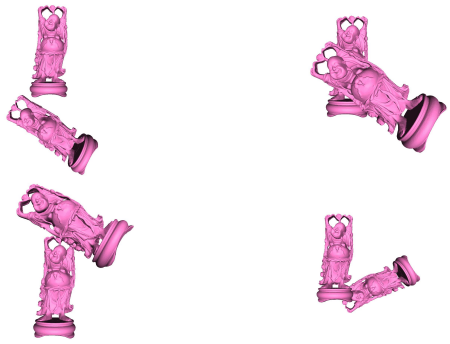


Figure 1: Our sphere-method uses a fixed rotation for every cycle. The moving object is rotated around the fixed object. After a cycle is finished, the rotation is changed.

The Sphere-Method The main idea of this method is to reduce the time for finding possible configurations. To this end, the 3D search space is reduced to 2 dimensions by using spherical coordinates. Nevertheless, it might happen to miss some interesting configurations. Within this method, we place the moving object on a sphere around the fixed object. The sphere should be bigger than the required distance. In the next step, we move the object towards the fixed object on a straight line through the center of the sphere until we reach the required distance. Because there could be several points that match the required distance on the straight line, it is possible to miss some configurations. In addition to the higher speed of this method, it is possible to define the number of located configurations in advance, because every straight line leads to exactly one configuration (see Fig. 1).

At the end of this procedure, we have got a large number of configurations for a user specified number of object-object-distances. This has to be done only once as preprocessing step, even if we add another collision detection library to the set later, or if we move to other platforms.

3.2 Benchmarking

The bulk of the work has been done in the previous step. In order to actually perform the benchmark, we just load the set of configurations. For each object-object distance we start the clock, set the transformation matrix of the moving object to all the configurations associated with the distance, and perform a collision test for each of them. After that, we can compute an average collision detection time for this distance.

3.3 Distance Computing

One method to determine the distance between two objects is to use the collision detection algorithms itself. We can build an offset object from the fixed object

 ISBN 978-80-86943-98-5

where the offset equals the specified distance. Then, we can conduct a binary search until we find a point where the moving object is just touching the offset object. However, offset objects can get very large for complex objects.

That is why we propose another method: The PQP-library offers the possibility to compute the distance between two objects by using swept spheres. With a given distance, we can also do a binary search until we find a point which matches the specified distance.

However, distance computing is more complicated than collision detection. Thus, this method is more time consuming. On the other hand, it is more accurate and less memory intensive than the offset object method. Therefore, we prefer this method for our benchmark. Another advantage of this method is that we know the exact distance between the objects during the binary search. We can use this information to delete cells in the grid method with a higher distance than the specified one. This accelerates the search for configurations.

Indeed, our benchmarking suite supports both methods for distance computing, because PQP is not Open Source software and, therefore, it is not possible to deliver it directly with our benchmarking suite.

Another problem that arises during distance computation concerns numerical stability. Because we are forced to floating point accuracy, it is not possible to find configurations with an exact distance while doing binary search. On account of this, we use an accuracy of 0.001% relative to the size of the fixed object in our benchmark. Of course, this accuracy can be changed by the user.

4 IMPLEMENTATION

Most collision detection libraries use proprietary internal data structures for data representation. Therefore, it is not possible to pass all kinds of objects directly to the algorithms. We chose OpenSG, a freely available scenegraph system for object management, because it offers support for many file formats, it is portable to many operating systems and, its data structures are well documented and easy to use. We wrote a wrapper for every collision detection library in order to convert the OpenSG data to the specific required data structures of the collision detection libraries. During initialization, our benchmark simply checks if the dynamically linked libraries are available and, if so, loads them.

We tested a wide variety of freely available collision detection libraries, precisely:

V-Collide: V-Collide, proposed by [HLC+97], is a wrapper with a simple interface for I-Collide and the RAPID library. In a first step, a sweep-and-prune algorithm is used to detect potentially overlapping pairs of objects. In a second step, the RAPID library is used for the exact pairwise test between a pair of objects. It uses an oriented bounding box test to find possibly colliding pairs of triangles.

PQP: PQP [GLM96] [LGLM99] is also based on the RAPID library. As with RAPID, PQP uses oriented bounding boxes. Furthermore, PQP is also able to compute the distance between the closest pair of points. For distance and tolerance queries, a different BV type, the so-called swept spheres, is used.

FreeSolid: FreeSolid, developed by [vdB99], uses axis-aligned bounding boxes (AABBs) for collision detection. For a fast collision test between the AABB hierarchies, the acceleration scheme described in [vdB97] is used. FreeSolid could also handle deformations of the geometry.

Opcode: Opcode, introduced by [Ter01], is a collision detection library for pairwise collision tests. It uses AABB hierarchies with a special focus on memory optimization. Therefore, it uses so-called no-leaf, i.e., BVHs of which the leaf nodes have been removed. For additionally acceleration it uses primitive-BV overlap tests during recursive traversal, whereas all other libraries described in this paper only use primitive-primitive-tests and BV-BV-tests. Like Freesolid, Opcode also supports deformable meshes.

BoxTree: The BoxTree, described in [Zac95], is a memory optimized version of the AABB trees. Instead of storing 6 values for the extents of the boxes, only two splitting planes are stored. For the acceleration of n-body simulations, the libraries offers support for a grid.

Dop-Tree: The Dop-Tree [Zac98] uses discrete oriented polytopes (k-DOPs, where k is the number of orientations) as BVs. k-DOPs are a generalization of axis aligned bounding boxes. The library supports different numbers of orientations. In [Zac98] it is shown that $k = 24$ guarantees the highest performance. Therefore, we also chose this number for our measurements. The set of orientations is fixed. This library also supports n-body simulation via grids.

When running the configuration space exploration (see section 3), the user simply specifies the objects he wants to test, the size of the grid, if he wants to use the grid-method or a step size for the spherical coordinates of the sphere-method. Moreover, a step size for the rotation of the moving object must be given and, finally, a distance. Then, our benchmark automatically generates a set of sample points for these specified parameters and benchmarks all available algorithms. It measures the times with an accuracy of 1 msec. Moreover, our benchmarking suite also offers scripts for the automatical generation of diagrams to plot the results of the benchmark.

5 RESULTS

Besides the distance between the objects, the performance of collision detection libraries mainly depends on the complexity and the shape of the objects. We used

 ISBN 978-80-86943-98-5

Figure 2: Some of the objects we used to test the collision detection libraries: A model of a castle, a helicopter and a laurel wreath

20 different objects in several resolutions in order to cover a wide range of use cases. All of the objects are in the public domain and can be accessed on our website. In particular, we used models of the Apollo 13 capsule and the Eagle space transporter, because they are nearly convex but have a lot of small details on the surface. To test the performance of the libraries on extremely concave objects we chose models of a helicopter, a luster, a chair, an ATST-walker and a set of pipes. Moreover, we used a laurel wreath to test intricate geometries. A Buddha model, a model of the Deep Space 9 space station, a dragon, and the Stanford Bunny were tested as examples of very large geometries. A model of a castle consists of very small, but also very large triangles. We used it to test the performance at unequal geometries. Accurate models of a Ferrari, a Cobra, and a door lock represent typical complex objects for industrial simulations. Finally, synthetic models of a sphere, a grid, a sponge, and a torus were used. Figures 2 and 3 show some of these objects.

Within our benchmarks, we simply tested a model against a copy of itself. However, our benchmark also supports the use of two different objects, but the first method is sufficient to make conclusions about the performance of the libraries.

We tested the libraries on a Pentium D CPU with 3 GHz and 1 GB of DDR2-RAM running Linux. All source code was compiled with gcc 4.0.2. We used the sphere-method with PQP for distance computing. We chose a step size of 15° for the spherical coordinates and a step size of 60° per axis for the rotations of the objects. With these values, we generated a set of 38000 sample configurations for every distance. We computed sample configurations for distances up to 40% of the object size in 1% steps, because in all example cases, there was no significant time spent on collision detection for larger distances. All these configurations can be downloaded from our web site[2].

The first reasonable finding of our measurements is that those algorithms, which use the same kind of BVH, behave very similar. Our second finding is that all algorithms have their special strength and weakness in different scenarios. E.g., the AABB-based algorithms like FreeSOLID, Opcode and the BoxTree were very well suited for regular meshes like the grid or the lustre, but also for meshes with very unequal triangle sizes, like the castle (see Fig. 7). In these cases, they were up to 4 times faster than the OBB-based libraries or the Dop-Tree. This is because in these test cases, AABBs fit the objects very well and therefore, the algorithms can benefit from their faster collision check algorithm.

When we used extremely concave and sparse objects, like the lustre or the ATST, or objects with lots of small details, like the Apollo capsule, the situation changed completely and the OBB-based algorithms, namely PQP and V-Collide, performed much better than the AABB-based libraries (see Fig. 5). This is, because with these kinds of objects, a tight fitting BVH seems to gain more than a fast BV test.

A special role played the Dop-Tree which combines the fast BV tests of the AABB-based algorithms with the tight BVs of the OBB-based libraries. As expected, this BVH is placed between the other two kinds of algorithms in most of the test scenarios.

Another interesting aspect we wanted to benchmark is the dependency on the complexity of the objects. Therefore, we tested all our models in different resolutions. The surprising result was, that there was no general dependency on the complexity for the algorithms we tested. E.g., in the lustre scene, the times increase nearly linearly with the number of polygons, for the AABB-based libraries, whereas it is nearly constant for the OBB-based algorithms. In the grid scenario, the increase is about $O(n \log n)$ for all algorithms (see Fig. 6). In the castle scene, the collision detection time seems to be independent from the complexity and in the chair scene, the collision detection time decreased for all algorithms with an increasing object complexity (see Fig. 7).

Summarizing, there is no all-in-one device suitable for every purpose. Every algorithms has its own strength in special scenarios. Therefore, the users should check their scenario carefully when choosing a special collision detection algorithm. A good compro-

[2] http://cg.in.tu-clausthal.de/research/colldet_benchmark/

 ISBN 978-80-86943-98-5

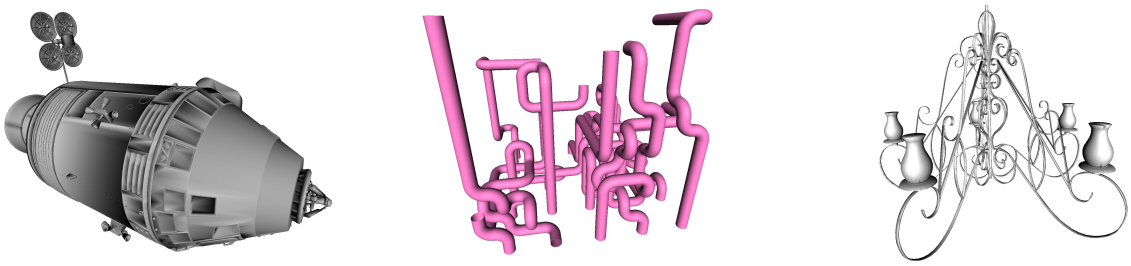


Figure 3: Some more of the test objects: A model of the Apollo 13 capsule, a set of pipes and a lustre.

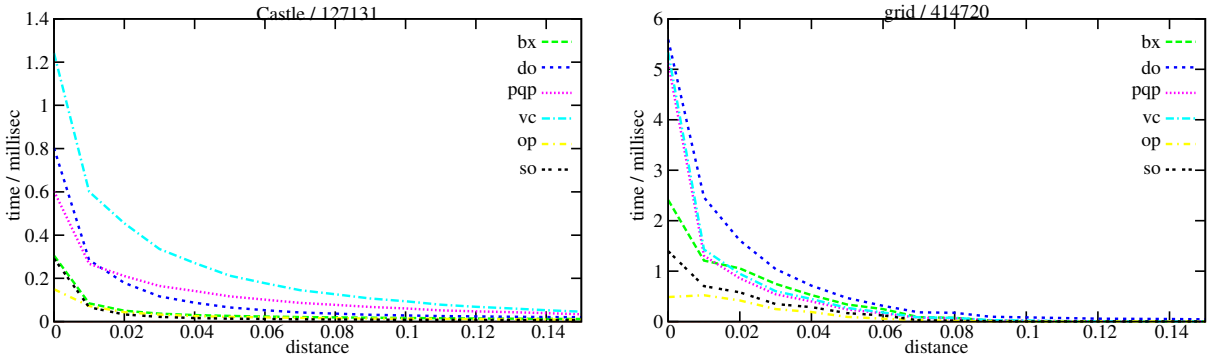


Figure 4: The results of the benchmark for the castle scenario in resolutions with 127 131 vertices and in the grid scene with 414 720 vertices. The x-axis denotes the relative distance between the objects, where 1.0 is the size of the object. Distance 0.0 means that the objects are almost touching but do not collide. The abbreviations for the libraries are as follows: bx=BoxTree, do=Dop-Tree, pqp=PQP, vc=V-Collide, op=Opcode, so=FreeSOLID. The AABB-based algorithms perform best in this kind of scenarios.

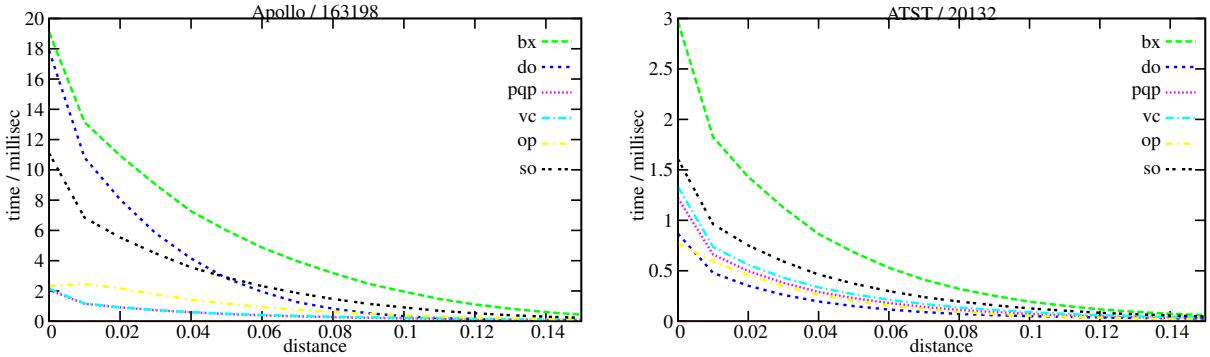


Figure 5: The results of the benchmark for the Apollo capsule with 163 198 vertices and the ATST walker with 20132 vertices. In these test cases, the OBB-based algorithms are much faster than the AABB-based libraries.

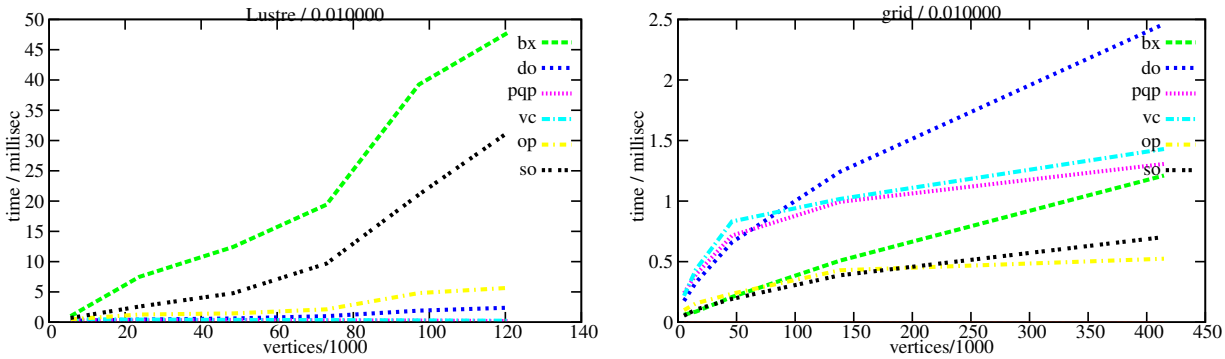


Figure 6: The results of the benchmark for the lustre scene and the grid scene for a distance of 1% relative to the object size. The x-axis denotes the number of vertices divided by 1000. The time for collision detection in the lustre scene increases nearly linearly for the AABB-based algorithms, whereas it seems to increase in $O(n \log n)$ for all algorithms in the grid scene.

 ISBN 978-80-86943-98-5

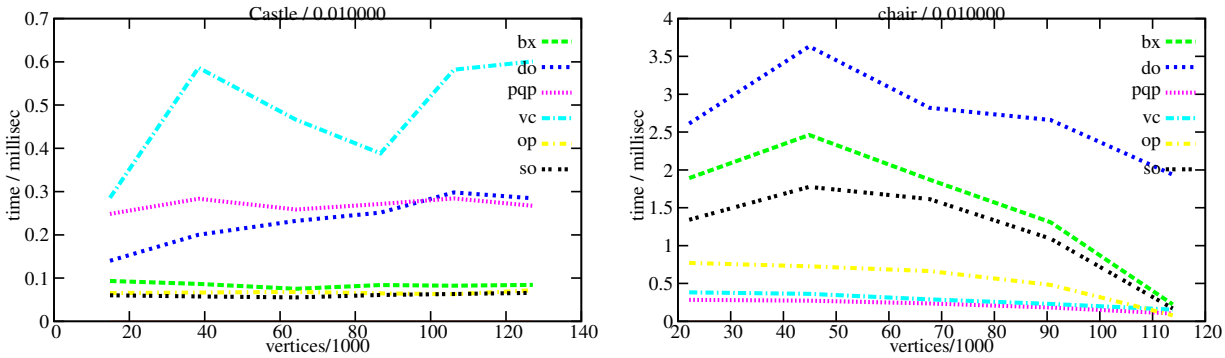


Figure 7: The dependency of the collision detection time from the complexity of the models in the castle and the chair scenes. The distance is fixed to 1% of the object size. In the castle scene, the collision detection time seems to be independent of the complexity, while in the chair scene, the time for collision detection even decreases with increasing complexity.

mise seems to be the Dop-Tree, because it combines tight BVs with fast BV tests. Moreover, in some cases, it could be helpful to increase the complexity of the model in order to decrease the time for collision detection, but this does not work in all cases. However, in nearly all test cases, all libraries are fast enough to perform real time collision checks even for very complex objects.

6 CONCLUSIONS

We presented an easy to use benchmarking method and a representative suite for benchmarking objects for static collision detection algorithms for rigid objects. Our benchmark is robust, fast, flexible, and it is easy to integrate other collision detection libraries. We used our benchmarking suite to test several freely available collision detection libraries with a wide variety of objects.

Our benchmarking suite is helpful for users to figure out the best fitting collision detection scheme to meet their specific requirements. The comparison of several algorithms yields a simple rule for choosing the optimal algorithm.

In the future, we plan to extend our benchmarking suite also for penetrating objects. Therefore, we will have to determine the penetration depth of a pair of objects. Another promising future project is the design of a benchmarking suite for more than 2 objects and for continuous collision detection algorithms. Moreover, a standardized benchmarking suite for deformable objects is still missing and could be very helpful for users.

REFERENCES

[CRM02] Stefano Caselli, Monica Reggiani, and M. Mazzoli. Exploiting Advanced Collision Detection Libraries in a Probabilistic Motion Planner. In *WSCG*, pages 103–110, 2002.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[HLC+97] Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. V-COLLIDE: Accelerated Collision Detection for VRML. In *VRML 97: Second Symposium on the Virtual Reality Modeling Language*, Rikk Carey and Paul Strauss, Eds. ACM Press, New York City, NY, 1997.

[LGLM99] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. In *Technical Report TR99-018*, 1999.

[OL03] Miguel A. Otaduy, and Ming C. Lin. CLODs: Dual Hierarchies for Multiresolution Collision Detection. In *Symposium on Geometry Processing*, pages 94–101, 2003.

[Ter01] Pierre Terdiman. Memory-optimized bounding-volume hierarchies. 2001. `http://www.codercorner.com/Opcode.htm`.

[vdB97] Gino van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools: JGT*, 2(4):1–14, 1997.

[vdB99] Gino van den Bergen. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects. *Journal of Graphics Tools: JGT*, 4(2):7–25, 1999.

[Zac95] G. Zachmann. The BoxTree: Exact and Fast Collision Detection of Arbitrary Polyhedra. In *SIVE Workshop*, pages 104–112, July 1995.

[Zac98] Gabriel Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, pages 90–97. Atlanta, Georgia, March 1998.

 ISBN 978-80-86943-98-5

Triangulation of uniform particle systems: its application to the implicit surface texturing

Florian Levet Xavier Granier Christophe Schlick

IPARLA Project - INRIA/LaBRI - Université Bordeaux 1

{levet,granier,schlick}@labri.fr

ABSTRACT

Particle systems, as originally presented by Witkin and Heckbert [32], offer an elegant solution to sample implicit surfaces of arbitrary genus, while providing an extremely regular distribution of samples over the surface. In this paper, we present an efficient technique that uses particle systems to rapidly generate a triangular mesh over an implicit surface, where each triangle is almost equilateral. The major advantage of such a triangulation is that it minimizes the deformations between the mesh and the underlying implicit surface. We exploit this property by using few triangular texture samples mapped in a non-periodic fashion as presented by Neyret and Cani [16]. The result is a pattern-based texturing method that maps homogeneous non-periodic textures to arbitrary implicit surfaces, with almost no deformation.

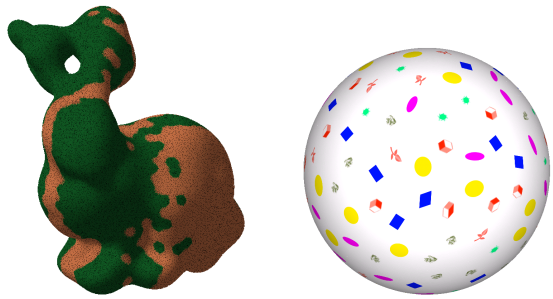
Keywords

Geometric Modeling, Implicit Surfaces, Particle Systems, Texturation

1 Motivation

Starting from the seminal work by Blinn [2] at the beginning of the eighties, implicit surfaces have been recognized as an elegant and powerful representation for 3D modeling. Although the several hundreds of paper that have been published on the subject during the last 25 years, a standard 3D modeling software environment that uses implicit surface as a general geometric paradigm is still not common practice in industry [18, 24]. So even though much work has shown that implicit surfaces combine different mathematical properties that make them very appealing, they are not widely used for real-scale applications in the computer graphics industry. There are mainly two reasons that may explain such a rather confidential use. First, implicit surfaces are not well adapted to the rendering pipeline used in current graphics hardware. Indeed, a rather expensive tessellation step is required to convert the surface into a polygonal mesh that can be efficiently processed by the hardware. Second, conventional texture mapping based on 2D texture patterns, which is by far the most ubiquitous solution to visually enhance geometric models, is really awkward to apply to implicit surfaces as there is no underlying parametrization.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency–Science Press, Plzen, Czech Republic.



(a) (b)

Figure 1: Pattern-based texturing of two different models.

Of course, 3D texturing techniques, such as solid textures [19, 34, 21] or octree textures [1, 12] can always be applied to implicit surfaces as they do not require any 2D parametrization. 3D textures represent a powerful solution to generate visual appearance of objects carved out from heterogeneous material such as marble or wood, for instance. Unfortunately, the visual complexity of most real life objects is much better simulated by 2D pattern-based texture mapping rather than 3D texturing. The main reason is that the normal

 ISBN 978-80-86943-98-5

vector of 3D textures does not align with the normal of the underlying surface.

Several authors have previously addressed the problem of tuning conventional 2D pattern-based texture mapping techniques for implicit surfaces. Most of these approaches will be briefly recalled in Section 2. The technique presented in this paper belongs to this stream of work. More precisely, the basic idea of this paper is to use the intrinsic features of uniform particle systems to drive a homogeneous non-periodic texture mapping on implicit surfaces with almost no deformation. The process is to rapidly generate a set of particles, apply a limited number of relaxation steps to achieve uniform sampling, and triangulate this set by taking into account the specific characteristics of the sampling. The resulting mesh, where each triangle is almost equilateral, is then used to support the parametrization of a pattern-based texture mapping scheme, similar to the one presented by Neyret and Cani [16].

The remainder of the paper is organized as follows. Section 2 recalls some related previous work on sampling and texturing of implicit surfaces. Section 3 introduces our algorithm to rapidly generate an almost-uniform set of particles over an arbitrary implicit surface. Section 4 focuses on the triangulation of this particle set, while Section 5 shows how to use this triangulation to get a local parametrization of a texture mesh that is then applied to the surface. Section 6 presents several experimental results obtained with our texturing process, and finally, Section 7 concludes and presents some directions that we are currently investigating.

2 Previous work

2.1 Particle systems

The first time that particles were used to sample a surface was in a complete modeling tool based on oriented particle systems developed by Szeliski and Tonnesen [26], but in their work there was no underlying implicit surface. Figueiredo et al. introduced [5] a system to sample implicit surfaces using particles. The authors used a similar relaxation process that [26] in order to achieve a uniform distribution of the sample points which are then used to compute a polygonal approximation of the implicit surface. Turk [30] used a reaction-diffusion method in order to re-tile polygonal surfaces according to the curvature, to get more points in regions of high curvature.

Witkin and Heckbert [32] developed a powerful modeling application by defining an adaptive repulsion and a split/death condition so that particles could either split or be removed from the surface. The technique generates a nice isotropic sampling of the implicit surface. Unfortunately, the convergence of the algorithm is rather slow, even for moderately complex surface, despite the improvement later proposed by Hart et al. [8] for automatic and numerical differentiation of the underlying surface.

Most improvements of particle systems addressed the problem of adaptive sampling of implicit surfaces according to curvature. Crossno and Angel [3] extended the work of Witkin and Heckbert by estimating a different repulsion radius for each particle according to its curvature. Similarly, Rosch et al. [23] used curvature information to sample unbounded surfaces and singularities. Meyer et al. [15] introduced a new class of energy functions for distributing either uniform or non-uniform particles on implicit surfaces. Levet et al. [14] presented an anisotropic sampling of implicit surfaces that locally takes into account the direction and value of principal curvatures.

Other works have presented ideas for sampling implicit surfaces for animation [6]. Some particle systems have also been used to polygonize implicit surfaces [5] or as a comparative quality technique for polygonizing implicit surfaces [11]. Su and Hart [25] presented an object-oriented particle framework designed to rapidly create new particle systems and applications.

Recently, some researches have focused on generating first the particles and, then, performing the relaxation process in order to get a uniform distribution of the particles. So, Galin et al [7] have developed a technique which takes into account the characteristics of the BlobTree [33] in order to realize a very fast uniform sampling of implicit surfaces. Even if a little slower, the method presented by Levet et al. [13] allows to generate a fast sampling of an implicit surface in order to perform a limited number of relaxations. This method works for any type of implicit surface and manage to generate uniform and non-uniform particles.

2.2 Texture mapping on implicit surfaces

Several authors have previously addressed the problem of tuning conventional 2D pattern-based texture mapping techniques for implicit surfaces. Note that most of the approaches are based on particle systems.

Turk [29] proposed to use particle systems to generate natural pattern directly on a generic surface by using a reaction-diffusion process. This approach becomes very expensive for high frequency details that require a huge number of particles.

Pedersen [20] described a texturing method for implicit surfaces by defining a set of patches over the surface. Each patch acts as a local parametrization and can be textured and interactively manipulated. The drawback of this approach is to require laborious and

 ISBN 978-80-86943-98-5

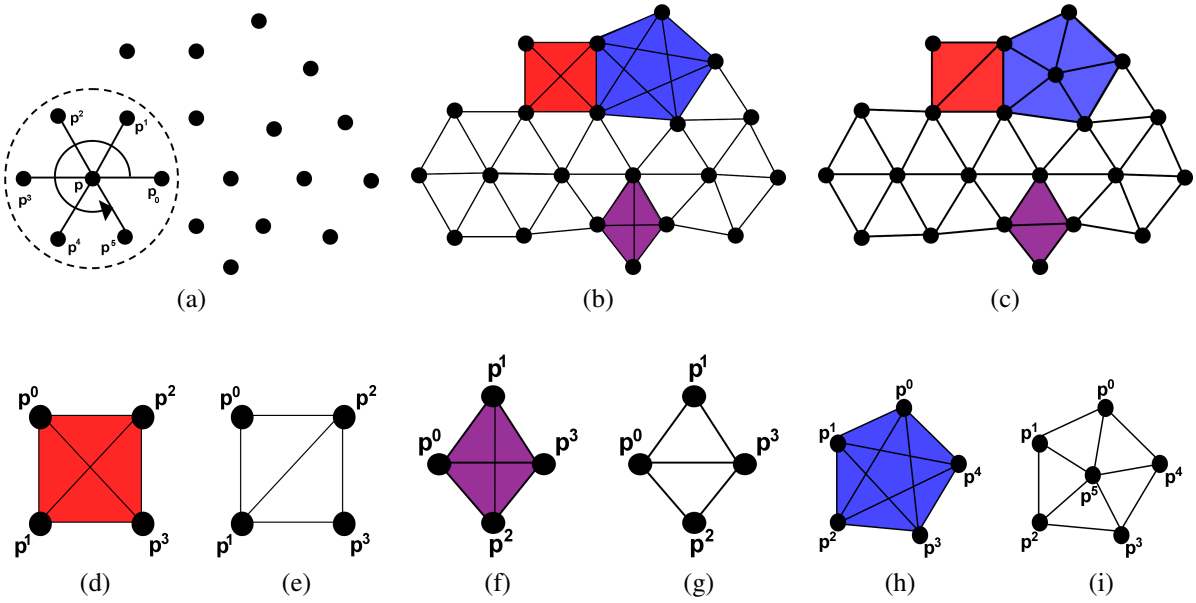


Figure 2: Triangulation of a uniform particle set.(a) Neighboring particles of P are ordered around it. (b) After the neighbors gathering some triangles intersect each others. (c) The simplification step discards all these triangles in order to create a good polygonization. The three problematic configurations are given in (d-f-h) while their solutions are given in (e-g-i).

sometimes tricky hand-tuning by the user to avoid deformation of 2D texture patterns.

Zonenschein et al. [35] proposed a simple texture mapping technique for implicit surfaces that uses a simple parametric support surface (e.g., sphere, cylinder). The mapping from the implicit surface to the support surface is done by a particle system. More precisely, particles are initialized on the implicit surface and then moved towards the support surface through an interpolation of the gradient fields of the two surfaces. While the original technique was rather limited, it has received several further extensions: to offer better control of local mappings [36], to account for composite implicit shapes [37] and for composite skeleton-based support surfaces [27, 28]. Despite this generalization, the technique is still not very robust in the case of implicit surfaces with complex topology or high-order genus.

3 Triangulation of the particle set

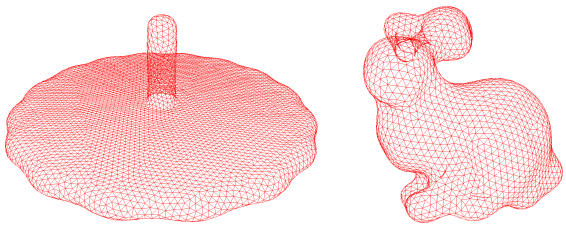
Crossno and Angel [4] have developed an algorithm that triangulates non-uniform particle set. Because it is not possible to predict the neighbors number of a particle, they have to deal with a lot of special configurations that makes their technique very complex to implement. Because we want to triangulate uniform particle systems, our algorithm is simpler.

The starting point of our technique is a set of uniform particles (which can be obtained by using the particle systems described in [32, 15, 13]). These uniform particle systems have some interesting characteristics. In the ideal scheme, each particle has six neighbors and the distance between each pair of them is $2r$ (r being the global repulsion radius for all the particles). The problem is that this ideal scheme can not be achieved easily. Indeed, as shown in [14], the detection of the convergence of particle systems is a difficult task. Thus, when the sampling of the implicit surface by the particles is stopped, each particle pair is not exactly at a distance of $2r$. Instead, the system has minimized the particle set energy in order that the average particle's number of neighbors is six and that the distance of each particle pair is around $2r$.

A naive solution would be to gather, for each particle, all the particles located at a distance of $2r$ and triangulate them. The problem is that, as said above, because of the last relaxation, the distance between neighboring particles can be a little more than $2r$. With this naive solution, we can thus miss some neighboring particles leading to uneven triangles. Besides, these configurations are difficult to identify. Our solution is to gather more particles than really needed because, in this case, the identification of the problematic configurations is easier. Thus, in our implementation, the following two-pass process is used: we first triangulate the particles with a simple neighboring search, and second, we simplify the mesh by discarding the intersecting triangles.

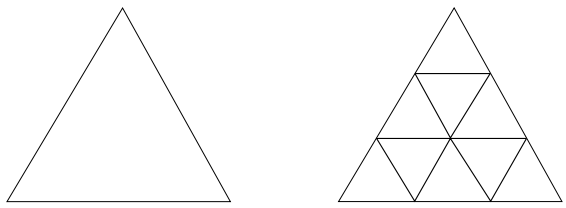
In the first step, we gather more neighboring particles than needed by using a $3r$ instead of $2r$ neighborhood, as it avoids some pathological configuration during the

 ISBN 978-80-86943-98-5



(a) (b)

Figure 3: Triangulation of two models.



(a) Original triangle. (b) Resulting tessellation.

Figure 4: Level 3 tessellation of a triangle.

simplification step. The gathered particles have then to be ordered around P (see Figure 2(a)). All the neighboring particles of P are projected on its tangent plane which allows us to do all the remaining computation in 2D. We select a neighbor of P as the reference particle P^r (in the Figure 2(a), the particle P^0 is defined as the reference particle P^r). For all the other ones, we compute the dot product between the reference vector $\overrightarrow{V^r} = \mathbf{p}^r - \mathbf{p}$ and the vectors $\overrightarrow{V^i} = \mathbf{p}^i - \mathbf{p}$, $\sum_{i=1}^{n}$. Finally, the ordering is done on the 2D cross product between $\overrightarrow{V^r}$ and $\overrightarrow{V^i}$. A simple triangle fan can now be created.

This results in a mesh with some intersecting triangles (see Figure 2(b)). Three different criteria are used to identify these triangles. First, for the configuration shown in Figure 2(d): $\mathbf{p}^3$ is not a neighbor of $\mathbf{p}^0$ but there is a triangle that links them. This configuration arises because we gather a $3r$ neighborhood instead of a $2r$ one. We can have now triangles between particles that are not neighbors. In order to identify such triangles, we have just to gather all triangles that have $\mathbf{p}^0$ as a vertex, and keep those that have $\mathbf{p}^3$ as well. Thanks to the characteristics of the particle systems, we are sure that the three other triangles exist ($(\mathbf{p}^0, \mathbf{p}^3, \mathbf{p}^2)$, $(\mathbf{p}^0, \mathbf{p}^1, \mathbf{p}^2)$, $(\mathbf{p}^3, \mathbf{p}^2, \mathbf{p}^1)$), and that we recover them. Once we have the four triangles, we just have to discard the $(\mathbf{p}^0, \mathbf{p}^1, \mathbf{p}^3)$ and $(\mathbf{p}^0, \mathbf{p}^3, \mathbf{p}^2)$ as P^0 and P^3 are not neighbors (see Figure 2(e)).

After having applied this first criteria, some intersecting triangles still remains. To identify them, we have to look at the neighborhood of a particle. This configuration arises when there is a triangle between three successive neighboring particles(see Figure 2(f)). Contrary to the configuration shown in Figure 2(d), P^0 and P^3 are neighbors. We still gather the four triangles, but this time we discard $(\mathbf{p}^0, \mathbf{p}^1, \mathbf{p}^2)$ and $(\mathbf{p}^1, \mathbf{p}^2, \mathbf{p}^3)$. Moreover, we have to cancel the neighborhood relation between P^1 and P^2 since they are not neighbors anymore (see Figure 2(g)).

The last configuration is slightly different: this time, the problem occurs because the particle set misses one particle. For instance, in the configuration shown in Figure 2(h), the ideal scheme is achieved by adding a particle at the star's gravity center. Thus, we discard all triangles that belong to the star, add a new particle P^5 located at its gravity center and generate all the corresponding triangles (see Figure 2(i)).

After the application of these three criteria, all the intersecting triangles (see Figure 2(b)) have been discarded leading to a polygonization with near-equilateral triangles. Figure 3 shows the results of our polygonization algorithm on two models. Table 1 shows the efficiency of our algorithm with various number of particles.

4 Tessellation of the triangulation

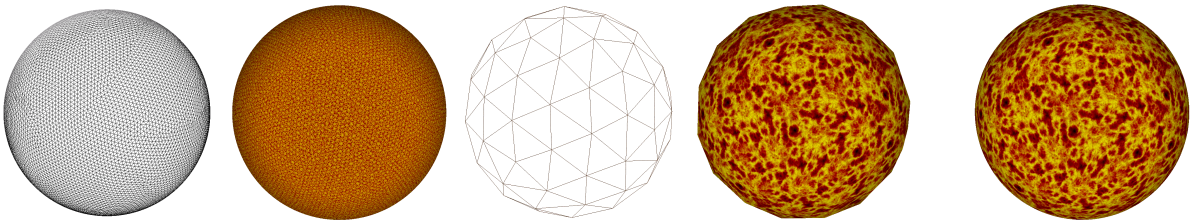
With the algorithm detailed above, we can rapidly generate a triangulation from a uniform particle set. So, the initial requirement is to efficiently get such a set of uniform particles. Unfortunately, because of their expensive relaxation process, particle systems proposed in the literature can not easily manage to sample a surface with thousands of particles in order to get the kind of triangulation shown in Figure 3.

As an alternative, we propose a two-step process, in which the implicit surface is first sampled with a small set of particles (this number is later increased by subdivision). The only need is that the general topology of the surface is not missed even if its geometric approximation is not sufficient because of the low number of particles. Thanks to the energy minimization of the particle system, the particles will be placed in order to catch as much details of the surface as possible. One nice characteristic of using the triangulation of the particle set given by the algorithm of Section 3 is that we still have the underlying implicit surface. Thus, we can further tessellate each triangle of the polygoniza-

# particles	Triangulation time	# triangles
682	0.010	1,360
2,813	0.045	5,622
17,178	0.263	34,292
36,464	0.685	72,915
73,379	1.175	146,671

Table 1: Times taken by the triangulation of uniform particle sets. All times are in seconds.

 ISBN 978-80-86943-98-5

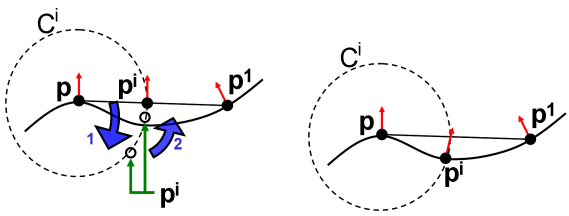


(a) High-detailed texture mesh (b) Direct visualization of the texture mesh (c) Low-detailed texture mesh (d) Direct visualization of the texture mesh (e) Visualization of the parametrized implicit surface

Figure 5: The scaling of the texture patterns is directly related to the texture mesh resolution. (a) Dense texture mesh (b) Texture patterns applied on dense texture mesh (high frequencies of the patterns are lost) (c) Coarse texture mesh (d) Texture patterns applied on coarse texture mesh (e) Coarse texture mesh mapping on the implicit surface (high frequencies of the patterns are preserved).

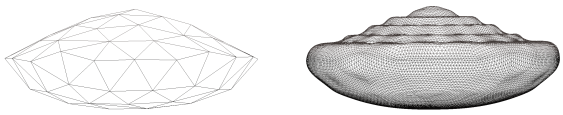
tion of the particles in order to get a good geometric approximation of the model.

Based on [9, 10], we subdivide each triangle as shown in Figure 4. But, contrary to them, we do not reproject the newly inserted vertices on the implicit surface by applying one step of the Newton-Raphson iteration. Indeed, we decided to reproject them by using an improvement of the geodesic reprojection of [13]. The idea is to reproject a newly inserted vertex $\mathbf{p^i}$ on the geodesic of the circle C^i centered on $\mathbf{p}$ (an original vertex of the triangle) which has a radius of $|\mathbf{p} - \mathbf{p^i}|$. But, contrary to [13], we use a dichotomic process. Indeed, because the evaluation of the implicit surface enables us to know if the vertex is inside or outside it, we can apply the iterative process shown in Figure 6(a). At the beginning, the vertex $\mathbf{p^i}$ is outside the surface. During the first reprojection step, a $\pi/4$ rotation is applied on $\mathbf{p}^i$ and the implicit equation is evaluated at this new position. If the vertex is still outside we do not change the direction of rotation. But, if the vertex is now inside, this direction of rotation is inverted (see Figure 6(a)) and its angle is divided by two. This process is repeated until the new vertex lies on the surface (see Figure 6(b)) according to a user-provided threshold on the implicit equation.



(a) (b)

Figure 6: Geodesic reprojection of a vertex: a dichotomic process is used. (a) Successive iterations of the reprojection are applied. (b) Final position of the vertex.



(a) (b)

Figure 7: Example of the tessellation of a model. (a) The implicit surface was sampled with a very low number of particles (74) leading to a polygonization with big triangles. (b) After applying a level 15 tessellation (the final model has 30,600 triangles), all the details of the surface are visible.

5 An application: the texturation of implicit surfaces

Neyret and Cani [16] have designed a pattern-based texturing of triangular meshes. Its principle is to create, over an existing arbitrary mesh, a *triangular texture mesh* composed of a small number of almost-equilateral triangles. The original mesh is used as a local parametrization of each triangle of the texture mesh. Patterns are then mapped on the texture mesh and the parametrization in a non-periodic way with no visible repetition of the patterns.

Thanks to the characteristics of our polygonization and tessellation, this technique can be directly adapted to the texturation of implicit surfaces. Indeed, the polygonization of the particle set (see Section 3) can be used as the texture mesh since each triangle is quasi-equilateral. Nevertheless, care must be taken on the size of the triangles (and thus the number of particles which sample the implicit surface). Indeed, patterns are mapped on the texture mesh. So, a large number of particles generated a good geometric approximation of the surface, but dramatically reduces the size of texture patterns which becomes hardly recognizable (see Figure 5(a)-(b)). On the other hand, a small number of particles, generates recognizable patterns, but does not provide a good geometric approximation of the surface (see Figure 5(c)-(d)).

 ISBN 978-80-86943-98-5

Triangulation, tessellation and texturation of different models							
Model	# particles	Triangulation Time in sec.	# triangles	Tessellation		Total # of triangles	Total Time
				Level	Time		
Sphere	76	0.002	148	15	0.344	33,300	0.346
Igea	117	0.002	230	10	1.545	23,000	1.547
Rabbit	257	0.004	509	5	0.708	12,725	0.712
Bunny	1,216	0.017	2,425	5	3.358	60,625	3.375

Table 2: Times for the texturing of different implicit surfaces with changing configurations (all given times are in seconds). We give the generation number of uniform particles which is the starting point of our algorithm. The number of triangles of the resulting texture mesh is shown as the time needed to triangulate and simplify this mesh. Finally, we give the time and the level of tessellation leading to the parametrization of these implicit surfaces.

The solution is to use the tessellation of the Section 4 as the parametrization of the texture mesh [16]. The patterns are then mapped from the texture mesh to the triangles resulting of the tessellation. Thus, the geometric density of the texture mesh can be improved as much as desired. The result of using the tessellation this way can be seen in Figure 5(e) which shows similar sized patterns as Figure 5(d), with the same geometric accuracy as Figure 5(b). This indirection between the texture pattern resolution and the geometric density of the texture mesh, offers a very flexible scheme to easily adjust the apparent size of the pattern on the implicit surface.

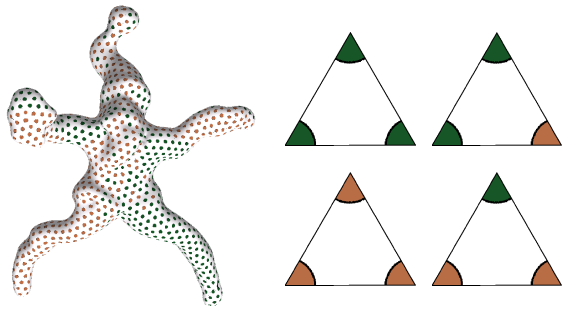
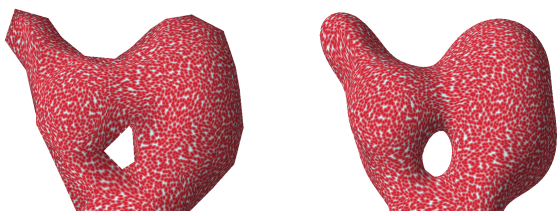


Figure 9: Texturing of the Santa model: the four patterns used (right) and the mapping result (left) are shown.

6 Results and discussion

All example models shown in this paper were built by using a PC with an Intel Pentium IV 3GHz processor and 1GB of main memory.

The sphere model shown in this paper was defined analytically, while the bunny, the Santa and the Igea models were reconstructed using Radial Basis Functions (RBF) [31]. These three models were reconstructed with a limited set of 200 to 500 initial points, because we used the original implementation of RBF with global support functions. But the technique can be easily adapted to more elaborated implicit reconstruction techniques, such as [17, 22] that reconstruct implicit surfaces from several millions of points. Note that our approach scales well since it only depends on the efficiency of the evaluation of the function that defines the implicit surface. This evaluation can be tuned using some efficient spatial subdivision structures. Note also that we removed some points that belong to the bunny ears, in order to increase the genus of the resulting surface, and thus better illustrate the genericity of our texturing technique (see Figure 8).



(a) Texture mesh. (b) Textured parametrization of the implicit surface.

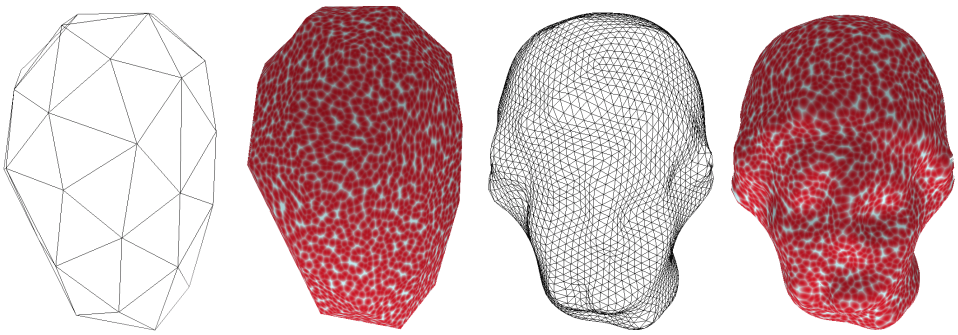
Figure 8: A closer view of the bunny ears.

Once the texture pattern has been selected, our technique allows the user to finely tune two important parameters. First, changing the radius of the particles provides a full control of the number of desired particles and, thus, of the apparent size of the resulting texture patterns over the implicit surface. Second, changing the tessellation level provides a full control of the geometric accuracy used by the rendering step of the textured surface.

In Table 2, we show some timings taken by our technique to texture implicit surfaces with various level of sampling and tessellation. With our method, simple models can be textured in less than a second without loosing detail of the underlying surface since they can be textured well with only a poor detailed texture mesh (Figure 5(e) shows the texturing of the model presented in Table 2). With more complex models as the bunny, the texture mesh must be a little more detailed in order to not miss topology of the model. So more particles are needed and the overall time of the texturing is larger.

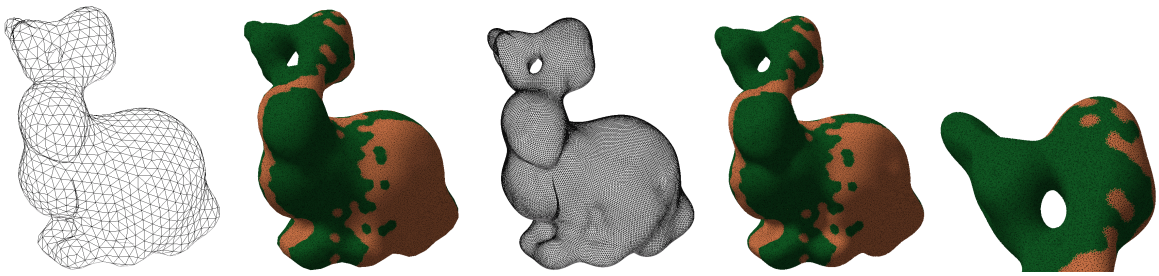
Figure 10 is a good example of our technique. Fig-

 ISBN 978-80-86943-98-5



(a) Texture mesh (b) Direct visualization of this texture mesh (c) Parametrization of the implicit surface (d) Texturing of this parametrization

Figure 10: Texturation of the Igea model. The texture mesh is very low-detailed. Note how the parametrization adds details on the model geometry with the same texture support.



(a) Texture mesh (b) Direct visualization of this texture mesh (c) Parametrization of the implicit surface (d) Texturing of this parametrization (e) Closer view of the bunny ears

Figure 11: Texturation of the bunny: the texture mesh and the parametrized implicit surface are shown. Note how our method manage to texture the bunny ears even if the genus of the model has been increased.

ure 10(a) shows the support texture mesh for the rendering of the implicit surface. It is composed of 74 triangles. At this time we can hardly recognize the Igea model. We give the parametrization of the implicit surface on which the texturing will be applied in Figure 10(c) (it's a level ten tessellation). In Figure 10(b) and Figure 10(d) the same texture mesh is used. But, while it is directly rendered in Figure 10(b), it is the parametrization of the implicit surface that is visualized in Figure 10(d) which enable us to recognize the Igea model.

7 Conclusion and future works

In this paper, we have presented a new very efficient algorithm to triangulate a uniform particle set. Based on the intrinsic characteristics of these uniform particle systems, our algorithm can triangulate more than 30,000 particles in less than a second. A second contribution of this work is an on-the-fly mesh refinement technique, driven by the underlying implicit equation, that allows to further tessellate each triangle of the polygonization. Finally, this triangulation combined with the tessellation can be directly used in order to map homogeneous non-periodic textures to arbitrary implicit surfaces thanks to a 2D pattern-based texturing method.

Based on this initial technique, we are currently investigating several extensions: for instance, the application of the method to generate out-of-core rendering of huge textured implicit surfaces reconstructed from scanned objects, as well as a local edition of the particle system in the case of animated skeleton-based implicit surfaces.

Acknowledgments

We wish to thank Fabrice Neyret for providing his pattern texture code for meshes.

References

[1] D. Benson and J. Davis. Octree textures. In *Proc. ACM SIGGRAPH '02*, pages 785–790, 2002.

[2] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[3] P. Crossno and E. Angel. Isosurface extraction using particle systems. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 495–498, 1997.

[4] P. Crossno and E. Angel. Spiraling edge: Fast surface reconstruction from partially organized sample points. In *VISUALIZATION '99: Proceedings of the*

10th IEEE Visualization 1999 Conference (VIS '99). IEEE Computer Society, 1999.

[5] L. H. de Figueiredo, J. Gomes, D. Terzopoulos, and L. Velho. Physically-based methods for polygonization of implicit surfaces. In *Proc. Graphics Interface '92*, pages 250–257, 1992.

[6] M. Desbrun, N. Tsingos, and M.-P. Cani. Adaptive sampling of implicit surfaces for interactive modeling and animation. In *International Workshop on Implicit Surfaces '95*, avril 1995. Published under the name Marie-Paule Gascuel.

[7] E. Galin, R. Allgre, and S. Akkouche. A fast particle system framework for interactive implicit modeling. In *SMI (Shape Modeling Internatinal)*, 2006.

[8] J. C. Hart, E. Bachta, W. Jarosz, and T. Fleury. Using particles to sample and control more complex implicit surfaces. In *Shape Modeling International 2002*, pages 129–136, 2002.

[9] T. Igarashi and J. F. Hughes. Smooth meshes for sketch-based freeform modeling. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 139–142. ACM Press, 2003.

[10] X. Jin, H. Sun, and Q. Peng. Subdivision interpolating implicit surfaces. *Computers & Graphics*, 27(5):763–772, 2003.

[11] T. Karkanis and A. J. Stewart. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics & Applications*, 21(2):60–69, 2001.

[12] S. Lefebvre, S. Hornus, and F. Neyret. *GPU Gems 2*, chapter Octree Textures on the GPU, pages 595–613. 2005.

[13] F. Levet, X. Granier, and C. Schlick. Fast sampling of implicit surfaces by particle systems. In *Shape Modeling International 2006 (Short Papers)*, 2006.

[14] F. Levet, J. Hadim, P. Reuter, and C. Schlick. Anisotropic sampling for differential point rendering of implicit surfaces. In *WSCG'2005*, pages 109–116, 2005.

[15] M. D. Meyer, P. Georgel, and R. T. Whitaker. Robust particle systems for curvature dependant sampling of implicit surfaces. In *Shape Modeling Internatinal 2005*, 2005.

[16] F. Neyret and M.-P. Cani. Pattern-based texturing revisited. In *Proc. ACM SIGGRAPH '99*, pages 235–242, 1999.

[17] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[18] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 2(8):429–446, 1995.

[19] D. R. Peachey. Solid texturing of complex surfaces. In *Proc. ACM SIGGRAPH '85*, pages 279–286, 1985.

[20] H. K. Pedersen. Decorating implicit surfaces. In *Proc. ACM SIGGRAPH '95*, pages 291–300, 1995.

[21] P. Reuter, B. Schmitt, A. Pasko, and C. Schlick. Interactive solid texturing using point-based multiresolution representations. In *WSCG'2004*, pages 363–370, 2004.

[22] P. Reuter, I. Tobor, C. Schlick, and S. Dedieu. Point-based modelling and rendering using radial basis functions. In *ACM Graphite 2003*, pages 111–118, 2003.

[23] A. Rosch, M. Ruhl, and D. Saupe. Interactive visualization of implicit surfaces with singularities. *Eurographics Computer Graphics Forum*, 16(5):295–306, 1997.

[24] R. Schmidt, B. Wyvill, M. Sousa, and J. Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. In *2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 53–62, 2005.

[25] W. Y. Su and J. C. Hart. A programmable particle system framework for shape modeling. In *Shape Modeling Internatinal 2005*, 2005.

[26] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. In *Proc. ACM SIGGRAPH '92*, pages 185–194, 1992.

[27] M. Tigges and B. Wyvill. Texture mapping the blobtree. In *Proc. International Workshop on Implicit Surfaces*, pages 123–130, June 1998.

[28] M. Tigges and B. Wyvill. A field interpolated texture mapping algorithm for skeletal implicit surfaces. In *Computer Graphics International*, pages 25–, 1999.

[29] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH '91*, pages 289–298, 1991.

[30] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, 1992.

[31] G. Turk and J. O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Georgia Institute of Technology, 1998.

[32] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proc. ACM SIGGRAPH '94*, pages 269–277, 1994.

[33] B. Wyvill, E. Galin, and A. Guy. The blobtree. warping, blending and boolean operations in an implicit surface modelling system. *Eurographics Computer Graphics Forum*, 18(2):149–158, 1999.

[34] G. Wyvill, B. Wyvill, and C. McPheeters. Solid texturing of soft objects. *IEEE Computer Graphics & Applications*, 7(12):20–26, 1987.

[35] R. Zonenschein, J. Gomes, L. Velho, and L. H. de Figueiredo. Texturing implicit surfaces with particle systems. In *ACM SIGGRAPH 97 Visual Proceedings*, 1997.

[36] R. Zonenschein, J. Gomes, L. Velho, and L. H. de Figueiredo. Controlling texture mapping onto implicit surfaces with particle systems. In *International Workshop on Implicit Surfaces '98*, pages 131–138, 1998.

[37] R. Zonenschein, J. Gomes, L. Velho, L. H. de Figueiredo, M. Tigges, and B. Wyvill. Texturing composite deformable implicit objects. In *SIBGRAPI'98*, pages 346–353, 1998.

 ISBN 978-80-86943-98-5

Efficient Region-Based Pencil Drawing

Shuo Sun
Tianjin, 300160
China, School of Science
Tianjin Polytechnic University
Sunshuo_0@163.com

Dongwei Huang
Tianjin, 300160
China, School of Science
Tianjin Polytechnic University
tjhuangdw@163.com

Abstract

This paper proposes an extension to the existing automatic pencil drawing generation technique based on Line Integral Convolution (LIC). The original LIC pencil filter utilizes image segmentation and texture direction detection techniques for defining outlines and stroke directions, and the quality of a resulting image depends largely on the result of the white noises and the texture directions. It may fail to generate a reasonable result when the white noises and the texture directions are not consistent with the texture structure of the input image. To solve this problem, we propose in this paper to improve the existed LIC-based method. First, a more accurate and rapid graph-based image segmentation method is introduced to divide the image into different regions. Second, we present a new region-based way to produce white noises and texture directions. We also demonstrate the enhanced LIC pencil drawing is closer to the real artistic style.

KeyWord pencil drawing, line integral convolution, image segmentation, Non-photo-realistic rendering

1. Introduction

Recently, Non-Photo-Realistic rendering (NPR) has become one of the most important research topics of computer graphics. A number of techniques have been developed to simulate traditional artistic media and styles, such as pen and ink illustration[Win94a, Win96a, Sal94a, Sal97a], graphite and colored pencil drawing[Tak99a, Sou99a, Sou99b], impressionist styles[Lit97a], paintings of various materials including oil[Her98a], water color[Cur97a] and so on. The existing researches on painterly image generation mainly take two different approaches. The first approach is to provide physical simulation to the materials and skills, and has been mainly combined with interactive painting systems or 3D non-photo-realistic rendering systems for generating realistic painterly images. The second approach is the painterly filtering, which involves taking an image and applying some kind of image processing or filtering techniques to convert it into an image of a painterly look. While many excellent painterly filtering techniques have been developed for generating brushstroke based paintings [Goo01a], relative few publications can be found on converting a source image into line stroke based drawings. In case of drawing, geometric information such as the outline of regions, the direction and shape of strokes becomes more critical, while it is usually difficult to extract such information from 2D raster images automatically. Instead of modeling line strokes geometrically, Mao etc [Mao01a] have developed a pencil drawing filter using Line Integral Convolution (LIC), a texture based flow visualization technique [Cab93a]. The technique utilized the similarity between the appearance of LIC images and pencil strokes, and succeeded in generating line stroke like images with pixel-by-pixel image filtering. It employs image segmentation and texture analysis technique to automatically detect the outlines and decide the stroke

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright UNION Agency – Science Press, Plzen, Czech Republic.

orientations for the images. Then for each region, if it contains directional textures, the texture directions are used as the stroke directions, otherwise a randomly chosen stroke direction is assigned. It may fail to generate a reasonable result when the white noises and the texture directions are not consistent with the texture structure of the input image. To solve this problem, we propose in this paper to improve the existed LIC-based method. First, a more accurate and rapid graph-based image segmentation method is introduced to divide the image into different regions. Second, we present a new region-based way to produce white noises and texture directions.

We realize automatic generation pencil drawing from 2D images. There are several advantages of our method comparing to the existed. First, according to the characteristics of the pencil drawings produced by the artists, different areas have different texture expressions. General speaking, some areas might have very complicated textures and others might be faint in textures, and that we are desire to achieve this kinds of artistic effects. Through controlling the noises of the different areas, we can not only express the shade areas of the texture but also hold some subtle area and necessary blank area in the image. The result is more similar to the artistic style.

The remainder of this paper is organized as the follows: Section 2 gives a short survey on related work. Section 3.1 introduces the original LIC-Based pencil filter method. Section 3.2 describes the algorithm graph-based image segmentation and Section 3.3-3.4 introduces our region-based pencil filter method. Section 4 concludes the paper and gives some examples.

2. Related Works

Pencil drawing has been an important topic since the beginning of painterly image generation research history. In an early 2D painting system called Pencil Sketch [Ver89a], a mouse based virtual tablet is provided for allowing users to interactively specify a set of parameters, such as the hardness of pencil, the pressure applied to a pencil, and the orientations of strokes. Recently, Sousa and Buchanan developed several pencil drawing rendering techniques based on an observation model of pencil drawings [Sou99a, Sou99b]. They built the models of pencil, paper and how lead pencils interact with drawing paper through a careful investigation of the real pencil drawings using scanning electron microscope. When the parameters of those models and the strokes are specified, a 2D image can be converted into a pencil drawing. 3D polygon models can be automatically rendered into pencil drawings by referring to the tone value lookup table for the parameter values of the models. Takagi and Fujishiro proposed to model the paper microstructure and color pigment distribution as 3D volume data and use volume ray-tracing for rendering color pencil drawings [Tak99a]. Other existing painterly image generation techniques closely related with our work are probably those successful works on pen–and-ink illustrations [Win94a, Win96a, Sal94a, Sal97a]. In their interactive systems, pen-and-ink illustrations can be generated either from 3D models or 2D images by using a set of pre-stored stroke textures. The largest difference between our technique and all these existing techniques is that our technique can generate a pencil drawing from a source image in a completely automatic way while all these existing techniques rely, to certain extent, on user interventions, for specifying the attributes and directions of strokes. Several commercial packages provide some filters for creating pencil drawing effects. For example, Jasc Paint Shop Pro software supports a black pencil filter. However, to obtain a satisfactory result with those filters, a user usually needs to combine the effects of many other filters and explore the best generation process experimentally through trial and error for many times.

3. LIC Pencil Filter

3.1. LIC Algorithm

Line Integral Convolution (LIC) is a texture based vector field visualization technique [Cab93a]. As shown in Figure 1, it takes a 2D vector field and a white noise image as the input, and generates an image which has been smeared out in the direction of the vector field through the convolution of the white noise and the low-pass filter kernels defined on the local streamline of the vector filed.

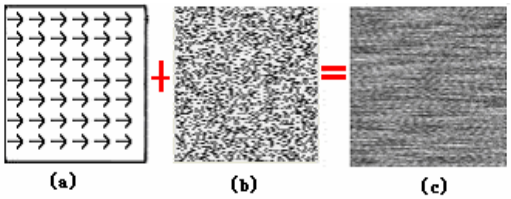


Figure 1 Line Integral Convolution (LIC) (a) Input vector field; (b) Input white noise; (c) Output result

The images in figure 2 show the basic algorithm of the LIC. The inputs are the vector fields and white noises.

 ISBN 978-80-86943-98-5

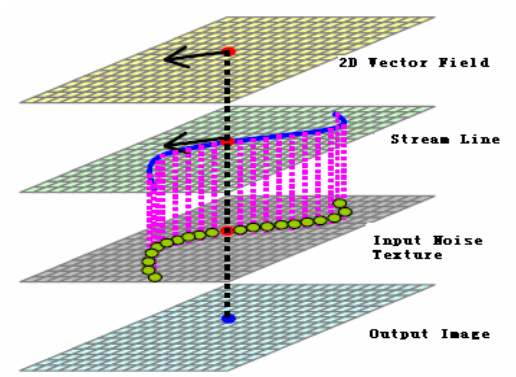


Figure 2 the basic algorithm of the LIC

P is the output pixel, $\rho(\tau)$ is stream line（$-L \le \tau \le L$），L is the half length of the stream line。$T(\rho(\tau))$ is the noise texture value in the stream line, $K(\tau)$ is a convolution kernel. So the pixel value in P is $T'(\rho(0))$:

$$T'(\rho(0)) = \frac{\int_{-L}^{L} k(\tau) T(\rho(\tau)) d\tau}{\int_{-L}^{L} k(\tau) d\tau}$$

The discrete form describes as follows: p_i is the discrete point in the stream line; W_i is the contribution of *pi* to P, namely the area that $K(\tau)$ cover between p_{i-1} with pi.

$$T'(P) = \frac{\sum_{i=0}^{N} T(p_i) W_i}{\sum_{i=0}^{N} W_i}$$

The idea of using LIC for pencil drawing generation was inspired by the visual similarity of LIC images and pencil drawings. As an LIC image is obtained by low-pass filtering a white noise along the local streamlines of a vector field, we can observe the traces of streamlines along which intensity varies randomly. Such traces have a similar appearance of pencil strokes where the variance of intensity is caused by the interaction of lead material and the roughness of paper surface. Figure 3(a) is a digitized sample of a typical imitative tone used in real pencil drawings. Figure 3(b), presents the very similar features as the tone image by LIC processing.

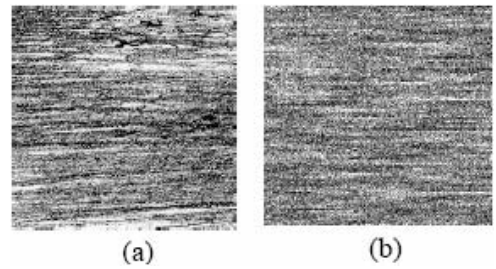


Fig.3: Comparison of a real pencil drawing (a) and an LIC texture (b).

3.2. The existed LIC pencil drawing method

In general, for producing a pencils drawing from a 2D source image, several steps are done. First we generate a white noise from the source image, then the original image is divided into different region and the boundary is extracted. Next we generate the vector field representing the orientation of strokes, and produce pencil drawing by applying LIC to the white noise and the vector field. Figure 4 depicts the algorithm of original LIC pencil filter. It converts a 2D source image into a pencil drawing in the following seven steps:

1. Generate a white noise (Figure 4(b)) from the source image (Figure 4(a)).
2. Segment the input image (Figure 4(a)) into different regions (Figure 4(c)).
3. Extract region boundary (Figure 4(d)).
4. Generate the vector field (Figure 4(e)) representing the orientation of strokes.
5. Generate stroke image (Figure 4(f)) by applying LIC to the white noise (Figure 4(b)) and the vector field (Figure 4(e)).
6. Add the boundary (Figure 4(d)) to obtain the drawing with outlines (Figure 4(g)).
7. Composite the resulting image (Figure 4(g)) with the paper sample (Figure 4(h)) to obtain the finished pencil drawing (Figure 4(i)).

 ISBN 978-80-86943-98-5

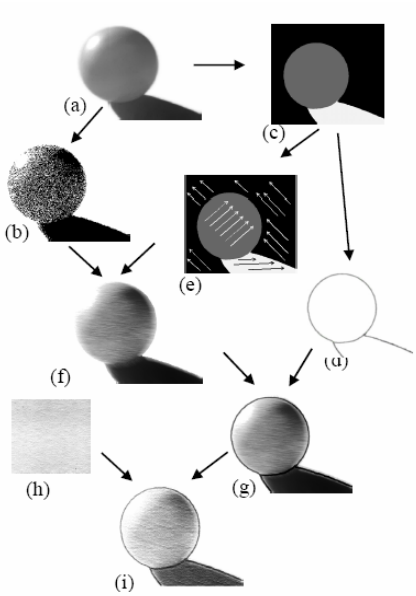


Figure4. The existed LIC algorithm (the image comes from [Mao01a])

4. Enhanced Region-based LIC Pencil Filter

4.1. Graph-Based Image Segmentation

A well used technique in pencil drawing for conveying the 3D shapes of objects and spatial relationship among different objects in a scene is to emphasize the boundary between two different regions by drawing outlines or changing the appearance of strokes in the two regions. To create such effect, we propose to divide the input image into different regions using existing image segmentation technique. In our current implementation, a Graph-Based image segmentation technique [Fel04a] is used for the region extraction. Contrasting to the method in [Mao01a], this method can dramatically promote the performance of our pencil filter. As shown in figure 5, the left image is segmented into many areas which represent the respective color distributions for each region.

Figure 5. Graph-Based image segmentation

Graph-based image segmentation techniques generally represent the problem in terms of a graph G = (V;E) where each node $v_i \in V$ corresponds to a pixel in the image, the edge set E is constructed by connecting pairs of pixels that are neighbors in an 8-connected sense (any other local neighborhood could be used). A weight is associated with each edge based on some property of the pixels that it connects, such as their image intensities. Neighbor edges are clustered into a forest, and each tree in the forest is related to a minimum spanning tree (MST). Finally each MST is a sub-area. To judge whether two trees can be merged into one tree, a predicate is defined. The predicate expression is showed as follow.

$$D(C_1,C_2)=\begin{cases} True\ if\ (Dif(C_1,C_2)>MinT(C_1,C_2) \\ False\ otherwise \end{cases}$$

$D(C_1,C_2)$ means the merging predicate of the areas of C_1 and C_2, $Dif\ (C_1, C_2)$ means the difference between the area C1 and the area C2. $MinT\ (C_1, C_2)$ is the minimum internal difference.

The advantage of the method is that the accuracy of the image segmentation can be adjusted by users and some details of the certain regions can be ignored. This will improve on the effect of the pencil drawing. In addition, this results in a graph with $O(n)$ edges for n image pixels, and an overall running time of the segmentation method of $O(n \log n)$ time.

4.2 Region-based Noise Production

The white noise image is generated in a way that the probability a white value is set for a pixel is proportional to the intensity level of the corresponding pixel in the input image. The gray-scale tone of a resulting pencil drawing is mainly decided by the white noise image. To match the tone between the input image and the resulting pencil drawing, we use the tone of the input image to guide the distribution of noise.

An important characteristic of the pencil drawing is its ability to preserve detail in low-variability image regions while ignoring detail in high-variability regions. The input image is then divided into many small regions which have corresponding meanings. The method mentioned in [Mao01a] dealt with the noise according to uniform criterion. The result noises would be failure to distinguish the important elements and the unimportant elements, when the range of the intensity of the image is small. Our region-based method solves the question through dynamic adjusting the threshold value of different areas. It is very important to pencil drawing. We simply introduce the algorithm.

 ISBN 978-80-86943-98-5

Because the processing of LIC is global, so some local regions with exquisite texture will be destroyed. In addition to, as a kind of sketch drawing, pencil drawing should keep some blank areas from the point of aesthetics. So we need to segment the original image into different region through above graph-based method, then we deal with the each area through giving certain threshold value. Commonly we select the average gray intensity of the certain regions as the threshold values.

Our method improves the producing way of white noise. We deal with the original gray image according to different gray intensity range in the respective regions. So the contrasts between light and shade in the results are more eminent and more similar to the artistic style. The formulation of white noise is showed as follows.

Let I_{input} be the intensity of a pixel in the input image, and P is a floating-point number generated with a pseudo-random function; R_i is the average intensity in the ith regions, and it is selected as the threshold to control the distribution of the noise. Then the intensity I_{noise} of the corresponding pixel in the noise image is decided in the following way:

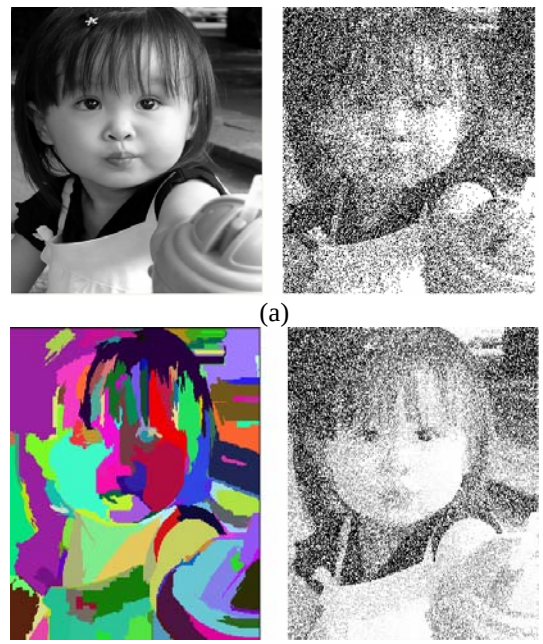
$$I_{noise} = \begin{cases} I_{noise1} = \begin{cases} \max 1 : (if(P > T_1)) \\ \min 1 : (else) \end{cases} (if(Input \le R_i)) \\ I_{noise2} = \begin{cases} \max 2 : (if(P > T_2)) \\ \min 2 : (else) \end{cases} (if(Input > R_i)) \end{cases} \quad (P \in [0,1])$$

Here max1 and max2 are the maximum gray values of the output pixels, which usually are 255; and min1 and min2 are the minimum gray values of the output pixels, which usually are R_i. However, we can adapt these values to fit for the whole tone of the pencil drawing.

$$T_1 = k_{i,1}(1 - \frac{I_{input}}{255}), T_2 = k_{i,2}(1 - \frac{I_{input}}{255})$$
$$k_{i,1} = \lambda_1(1 - \frac{I_{input}}{255}), k_{i,2} = \lambda_2(1 - \frac{I_{input}}{255})$$
$$\lambda_1, \lambda_2 \in [0,1]$$

According to our method, λ_1 and λ_2 are two experiential values. Our initial experiment result suggests that a default value of 0.7 and 0.3 produce a visually acceptable result for most scenes. We also allow users to interactively adjust the values of λ_1 and λ_2.

Through above method, the noise of each area can be controlled to match the features of the pencil drawing to a great extent. As shown in figure 6, the picture (a) show the noise produced by old way, and the picture (b) is produced by our method. Obviously, the result noise dealt with by our method can reflect the nature of the pencil drawing. Some areas need to be emphasized corresponding to more dense noises, and some areas need to be fade out corresponding to sparse noises.



(a)

(b)

Figure 6 The existed noise algorithm (a), and Region-Based noise method (b).

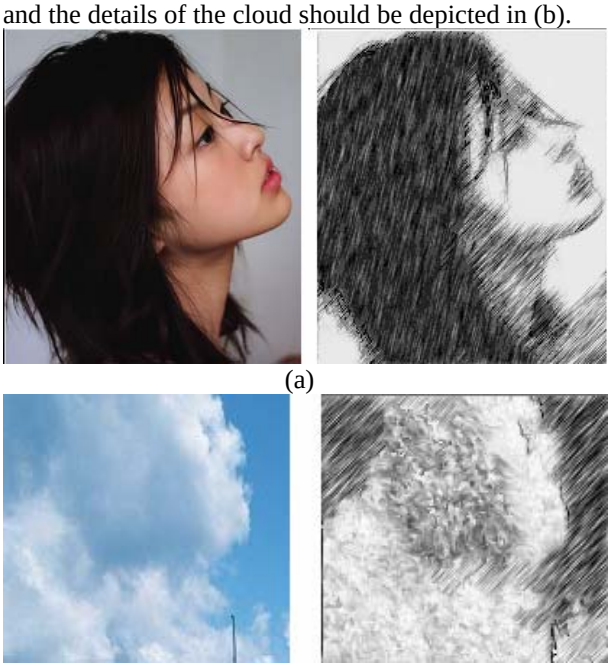
4.3.Stroke Orientation

When using imitative tone, the direction of strokes is an important factor contributing to the impression of a pencil drawing. The stroke direction is also important for conveying the shapes and textures of objects. If a texture presents a directional feature, then strokes should be oriented in a direction matching that of the original texture. For example, when we paint human hair, we would have our pencil follow the direction of hair strands. To produce such effect, Mao etc [Mao01a] implemented a Fourier texture analysis technique for extracting the local texture directions. The basic principle of the technique is that if a texture presents a specific direction, then after transform the texture into frequency domain, the power spectrum should have large values in the direction orthogonal to the direction of the texture. They deal with each pixel

 ISBN 978-80-86943-98-5

in the input image and compute respective texture direction by the analysis of the FFT.

The method mentioned above is very time-consuming, and they don't consider the local structure features of the image. Usually, some regions have similar directions of the strokes, and these areas can be dealt with the uniform direction according to artistic style. On the other hand, the directions of the strokes vary dramatically in some regions, so we keep the results of the respective directions, and then the details features in the area should be represented. As shown in figure 7, the direction of hair strands is important in (a) and the details of the cloud should be depicted in (b).



(a)

Figure 7 the direction of hair strands (a), and the details of the cloud (b).

Here we propose a method to produce the direction of the stroke according to the features in the regions. First, the orientation vectors of the pixels are calculated by sobel operator. Then a criterion is presented to decide the direction of the stroke.

Let $V_{i,j}$ $(j = 0,1,2,... m_i)$ be the field vector of the jth pixel in the ith area. It is computed by sobel operator. Here m_j is the numbers of the pixels in the jth area.

Then the mean of the field vectors in each area is calculated. At the same time, the CV (Coefficient of Variation) is computed to reflect the orientation's distribution.

$$V_i = (\sum_{j=0}^{m_i} V_{i,j}) / m_i$$

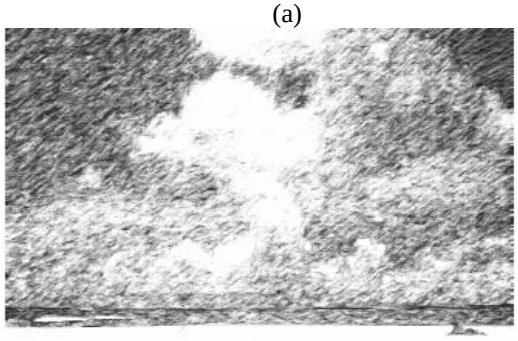
$$D_i = \sum_{j=0}^{m_i} (V_{i,j} - V_i)^2$$

The final field vector of the each pixel is determined through the following criterions:

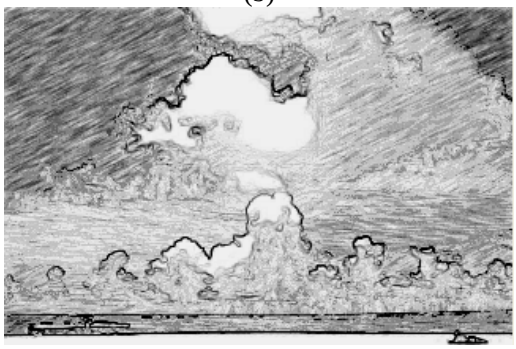
$$FV_{i,j} = \begin{cases} V_{i,j} : if\,(D_i / m_i > T) \\ V_i : (else) \end{cases}$$

Here T is a given threshold. Experiments suggest that a default value of 0.7 produce a visually acceptable result for most scenes.

(a)



(b)



(c)

Figure 8. The contrasts of the stroke directions

Figure 8 (a) is original image, and (b) is the result in [Mao01a]. The picture (c) is our result. Obviously,

 ISBN 978-80-86943-98-5

our result not only reflects the details in some areas but also embody the style of the pencil drawing.

5. Experiment results

We have implemented all the techniques described above and built an automatic pencil drawing generating system on Windows environment. Basically, after the input image is specified, a pencil drawing style image will be generated automatically with a set of default value and other information derived by the system from the input image. Users are also allowed to interactively specify some parameters, such as the parameters for controlling the region tone of the output image, the threshold used for graph-based image segmentation and the length of convolution.

Comparing to the existing method in [Mao01a], the new algorithm is much faster than the existing one and more similar with the artistic style of the pencil drawing. The existing LIC pencil filter took about 60 seconds to generate an image of size 350*280 on a Pentium IV PC. Our method only require about 5 seconds. For an image of size 1024*768, our method takes about 20 second, and the more important feature contrasting to the method in [Mao01a] is that our method dynamically analyzes the texture structure of the regions. So it is effective and lifelike. Several results of the pencil drawing as shown in figure 9.

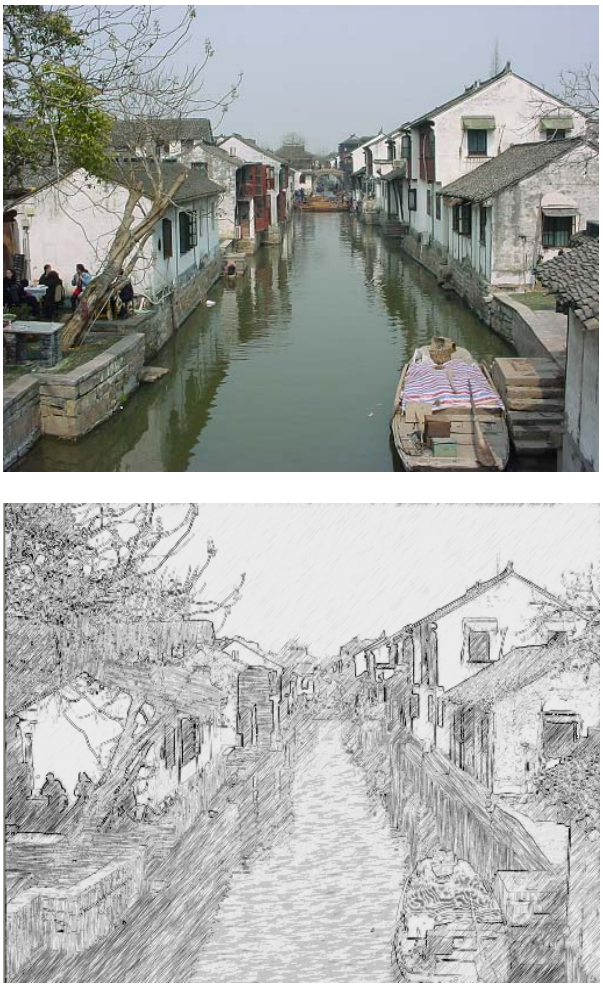


Figure 9. Several results of the pencil drawing

6. Conclusion

We propose in this paper to improve the existed LIC-based method. First, a more accurate and rapid graph-based image segmentation method is introduced to divide the image into different regions. Second, we present a new region-based way to produce white noises and texture directions. The new algorithm is much faster than the existing one and more similar with the artistic style of the pencil drawing. We believe the purpose of NPR systems is not to replace artists, but rather to provide a tool for users with no training in a particular medium. Potential application fields of our technique include producing posters from photo graphs, processing videos into pencil drawing style animations, obtaining the preparatory sketches for creating paintings of other styles, and so on.

 ISBN 978-80-86943-98-5

The major future research directions include the realization of abstraction and focusing, establishment of the curved strokes, extension to colored pencil drawing and application to non-photorealistic rendering of 3D models and scenes.

7. Acknowledgements

The work reported in this article is supported by the National Natural Science Foundation of China (No.10472077, No.50375107).

8. References

[Cab93a] B. Cabral and C. Leedom, “Imaging Vector Field Using Line Integral Convolution”, *SIGGRAPH93 conference Proceeding*, pages 263-270, 1993.

[Cur97a] C. J. Curtis, S. E. Anderson, J. E. Seims, Kurt W. Fleischer, and David H. Salesin, “Computer-Generated Watercolor”, *SIGGRAPH 97 conference proceeding*s, pages 421–430, 1997.

[Fel04a] P.F. Felzenszwalb, D.P. Huttenlocher “Efficient Graph-Based Image Degmentation” *International Journal of Computer Vision[J]*, 2004

[Goo01a] Gooch, *Non-Photorealistic Rendering*, AK Peters Ltd, 2001.

[Her98a] A. Hertzmann, “Painterly Rendering with Curved Brush Strokes of Multiple Sizes”, *SIGGRAPH 98 conference proceeding*s, pages 453–460, 1998.

[Lit97a] P. Litwinowicz, “Processing Images and Video for An Impressionist Effect”, *SIGGRAPH97 conference proceedings*, pp.407-414, 1997.

[Mao01a] X. Mao, Y. Nagasaka and A. Imamiya", Automatic Generation of Pencil Drawing from 2D Images Using Line Integral Convolution", Proceedings of the Seventh International Conference on Computer Aided Design and Computer Graphics CAD/GRAPHICS2001, PP. 240-248, 2001. *conference proceedings*, pages 138-143 , 1989.

[Sal94a] M.P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin, “Interactive Pen–And–Ink Illustration”, *SIGGRAPH 94 conference proceedings*, pages 101–108, 1994.

[Sal97a] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin, “Orientable Textures for Image-Based Pen-and-Ink Illustration”, *SIGGRAPH 97 Conference Proceeding*s, pages 401–406, 1997.

[Sou99a] M.C.Sousa and J.W.Buchanan, “Observational Model of Blenders and Erasers in Computer-Generated Pencil Rendering”, *Graphics Interface '99 conference proceedings*, pages 157-166, 1999.

[Sou99b] M.C.Sousa and J.W.Buchanan, “Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models”,*EUROGRAPHICS '99 conference proceedings*, pages 195-207, 1999.

[Tak99a] S. Takagi, I. Fujishiro and M. Nakajima, “Volumetric modeling of colored pencil drawing,” *Pacific Graphics'99 conference proceedings*, page 250-258, 1999.

[Tam84] H. Tamura Edited, “Introduction to Computer Image Processing (In Japanese)”, *Soken Shuppan Publisher Inc*. 1984, ISBN4-7952-6304-3.

[Ver89a] A. H. Vermeulen and P. P. Tanner, “Pencil sketch—A Pencil-Based Paint System”, *Graphics Interface '8.*,1989

[Win94a] G. Winkerbach, D. H. Salesin, “Computer-Generated Pen-and-Ink Illustration”, *SIGGRAPH94 conference proceedings*, pages 91-100, 1994.

[Win96a] G. Winkenbach and D. H, Salesin, “Rendering Parametric Surfaces in Pen and Ink”, *SIGGRAPH96 conference proceeding*s, pages 469–476, 1996.

Accelerating and enhancing rendering of realistic ocean scenes

E. Darles, B. Crespin, D. Ghazanfarpour
XLIM – Université de Limoges
83 r d'Isle, 87000 Limoges, France
[emmanuelle.darles|benoit.crespin|djamchid.ghazanfarpour]@xlim.fr

ABSTRACT

Modeling and rendering realistic ocean scenes has been investigated by many researchers in the past few years, providing different models of ocean waves, fine-scale details, and light-water interactions. This paper describes a practical data structure for accelerating ray-marching for a well-known wave model defined as a sum of trochoids waves. This data structure exhibits nice properties that allow to take into account spatial and temporal coherence as well as reducing aliasing effects. In addition to this, it offers a unified framework in which fine-scale details, such as scattering phenomena due to spray, foam or particles suspended in the water, can be added easily. Finally, we present some benefits of the model, which can handle physically-based light-water interactions and glare effects.

Keywords

Physically based modeling, ocean scenes, graphics data structures, ray-marching, sphere tracing

1. INTRODUCTION

Realistic ocean scenes are more and more present in computer-generated films, motion pictures and electronic games, but creating such scenes is a difficult task because a lot of modeling and rendering aspects need to be addressed. Furthermore, as the ocean surface is usually very large, raytracing is computationally intensive. The most important aspect in this context is the surface model used to represent realistic waves; it should allow realistic waves shape and motion at the lowest possible cost.

Remaining issues are light-water interactions and complex phenomena occurring at the surface, which usually depend on atmospheric conditions:

- foam, appearing through advection and turbulent diffusion phenomena during the simulation, modifies radiance at the surface
- sprays, *i.e.* small droplets created by foam interacting with wind, involve attenuation and diffusion of light above the surface
- second-order scattering: the attenuation and diffusion of light within the water body, due to particles in suspension

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

Taking these phenomena in consideration greatly enhances the realism of ocean scenes as long as they rely on physical parameters and are approximated with sufficient accuracy. In a raytracing environment we also wish to reduce computations and memory requirements.

This paper addresses all these issues, by proposing a new method based on a set of spheres that combines an efficient surface model and physically-based complex phenomena, integrated into a unique data structure. After reviewing different surface models and complex phenomena proposed in the literature in section 2, we choose a parametric model based on real measurements that gives realistic results and present an efficient method for computing ray-surface intersections based on sphere tracing in section 3. This approach is then extended to include complex phenomena as described above, using the same simple and efficient data structure (section 4). Finally, results are presented in section 5 along with considerations about implementation and performances of our approach.

2. RELATED WORK

Modeling and rendering realistic ocean scenes has been investigated by many researchers in the past few

 ISBN 978-80-86943-98-5

years. We focus mainly on papers dealing with scenes depicting the ocean far from the shore, i.e. where breaking waves do not appear. Real-time methods are also briefly reviewed although our work is more oriented towards ray-tracing. A more general state-of-the-art report can be found in [Igl04].

2.1 Ocean surface models

Models that attempt to represent the ocean surface accurately can usually be classified in two different approaches: parametric or spectral. Parametric approaches [FR86, Pea86, TB87, CGG01] represent the ocean surface as a sum of periodic functions which describe waves as a circular motion of water particles. This procedural model is very efficient but does not yield realistic results. On the other hand, spectral methods [MWM87,Tes01] are based on oceanographic measures, synthesized by spectral analysis and hence represent the ocean surface as a height field computed from a sum of sinusoids of various amplitudes and phases; small-scale waves and ripples are modeled directly by adding noise perturbation. This approach ensures high realism, but is not easily controllable.

To overcome these problems, hybrid, procedural models were proposed [PA01, TG02]. The parameters can be obtained automatically from real oceanographic spectra, which allows realistic animations of the ocean surface. Another advantage of the model is its implicit definition, which is well-adapted to ray-tracing. Real-time representations of hybrid models can also be obtained by generating an optimized surface mesh only where necessary, thus reducing both the sampling and the number of waves on the fly [HNC02, Mit05, HVT+06]. But real-time methods can hardly handle complex phenomena reviewed in the next section.

2.2 Light-water interactions and complex phenomena

The model proposed in [PA01] includes the apparition of foam on waves crests, which locally modifies the optical properties of the ocean. The surface is discretized and rendered by a Monte Carlo path tracer, which also takes into account the turbidity of the ocean, *i.e.* the opacity of the surface depending on particles suspended in the water. This parameter allows to determine the amount of light scattered towards the surface, and provides realistic simulations of several types of water. Second order scattering is also taken into account in [IDT03] by computing radiance at sampled points on a set of horizontal slices within the water volume using volume rendering graphics hardware.

Complex phenomena, such as foam, spray or splashes are usually modeled and rendered using particle systems [JG01,TFK+03,HW04]. In [TRS06], animation of drops and small-scale interactions such as ripples or splashes are obtained by combining a two-dimensional water surface simulation and a three-dimensional fluid simulation along with a physically based model.

As a conclusion, among these methods only one [PA01] seems able to provide realistic ocean scenes, by combining an efficient surface model and physically-based complex phenomena. Our work follows the same goal, by integrating these different aspects into a unique data structure.

3. RAY-WAVES INTERSECTIONS

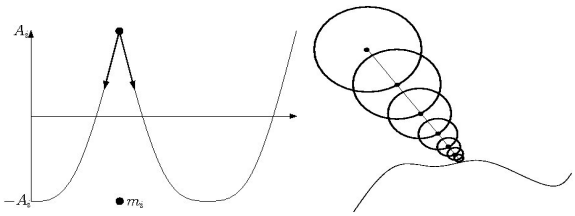


Figure 1. (a) Trochoid example (b) Sphere tracing

Our implementation uses the ocean surface model of [TG02], where the reader can find more details. The model is based on a summation of elementary waves called *trochoids* (see Fig. 1a), which yields realistic waves shapes varying from a smooth profile in the case of calm ocean, to a sharpened crested shape in case of an agitated ocean surface due to greater wind activity. Each trochoid is defined as a displacement function applied to a reference plane, parameterized by its amplitude A_i, its frequency f_i, its wavelength λ_i, its direction of propagation in the plane θ_i and its phase ϕ_i. Several methods are proposed in [TG02] to extract correct parameters from real measurements. All the pictures in this paper were obtained using Pierson-Moskowitz spectrum; high-frequency trochoids were amplified to represent capillary waves created by the wind.

Realistic ocean waves are obtained using 50 different trochoids or more. The counterpart is that a lot of evaluations of trigonometric functions (or access to a precomputed, lookup table storing trochoids) are needed to compute the displacement of a single point; this is particularly critical when rendering ocean scenes using ray marching, since it involves computing many intersections between rays originating from the camera and the ocean surface.

3.1 Original sphere tracing formulation

Consider the zero-set of an implicit function $F: R^3 \rightarrow R$, *i.e.* the set of points $p \in R^3$ that satisfy

 ISBN 978-80-86943-98-5

$F(p) = 0$. Brute-force ray marching attempts to find the intersection between a ray and such a surface using a fixed, constant increment Δ. Let p be a point along the ray such that $F(p) > 0$, and let $p' = p + \Delta$ be another point along the ray such that $F(p') < 0$; in that case, the ray crosses the surface somewhere between p and p'. Otherwise the process continues for p' and $p'' = p' + \Delta$. The choice of Δ is critical: if it is too small, the whole process requires a huge amount of evaluations of the implicit function, if it is too large intersections can be missed. A solution to this problem is the *sphere tracing* method [Har96, HW96], applicable for implicit surfaces that exhibit a Lipschitz bound L such that $|F(p)-F(q)| \leq L||p-q||$, $\forall\, p,q \in R^3$. In that case, the brute-force algorithm can be enhanced by computing Δ at each evaluation step by: $\Delta = F(p) / L$. This guarantees that the next point along the ray $p' = p + \Delta$ satisfies $F(p) \geq 0$, making it possible to converge very quickly to the intersection (see Fig. 1b). The term "sphere" refers to the fact that Δ represents the radius of a sphere guaranteed not to intersect the zero-set of the implicit surface.

Suppose that F is defined as a sum of N functions:

$$F(p) = \sum_{i=1}^{N} n_i(p) \qquad (1)$$

Unenhanced sphere tracing in that case would use the global Lipschitz bound L of function F; but this would also require to evaluate each individual term at every step along the ray. The solution is to maximize L by computing an "efficient" bound L_e such that $L_e < L$ without evaluating each term; this is summarized in Algorithm 1.

Require: components sorted by decreasing $n_i(p)/L_i$
Require: minimum m_i of each component
$L_e = 0;\ den_e = 0;\ num_e = m_0 + m_1 + ...$
for all component i in the sorted list
  $num_e \mathrel{+}= n_i(p) - m_i$; $den_e \mathrel{+}= L_i$;
  if $(L_e > (num_e / den_e))$ then **return** L_e;
  else $L_e = num_e / den_e$;

Algorithm 1: computation of L_e for a point p

It can be noted that the sorting step still seems to require the evaluation $n_i(p)$ of each component; actually an approximated value based on the previous iteration is used instead. Experiments using this method show speedups by at least a factor 10 compared to brute-force raytracing, for fractal surfaces defined by about 10 individual components.

3.2 Sphere tracing the ocean surface

In the case of our ocean surface model, and assuming that the equation defining the reference plane is $y=0$, the displacement n_i at a point $p:(x, y, z)$ at time t is defined as:

$$n_i(p) = ((y - A_t)/N) - A_i$$
$$trochoid(k_i(x * \cos\theta_i + z * \sin\theta_i) - \omega_i * t + \varphi_i) \qquad (2)$$

where N is the total number of waves, At is a global amplitude term, $\omega_i = 2\pi f_i$, $k_i = 2\pi / \lambda_i$ and *trochoid* describes the motion of a water particle. Thus, the implicit function has the same form as in Eq. 1.

In order to apply Algorithm 1, we still need to address several issues:

- ***choice of a minimum m_i for each component***: since each component is bounded, its lowest value occurs when $(y-A_t)/N=-A_i$ and the trochoid reaches its maximum value. In that case, the global minimum is given by $m_i=-2A_i$ (see Fig. 1a).
- ***choice of a Lipschitz bound L_i for each component***: in the case of a displacement function applied to a reference plane (such as a trochoid), we can simply use its maximum derivative. For our model, this maximum slope is obtained when the term $trochoid(k_i(x \cos\theta_i + z \sin\theta_i) - w_i t + \phi_i) = 1$
- ***antialiasing***: as in [TG02], a simple antialiasing technique consists in progressively reducing the amplitudes of high-frequency components depending on the projection of the corresponding pixels on the screen (see Fig. 2). In our case, unnecessary components are neglected during the computation of the global efficient bound L_e, thus reducing the overall computation cost.

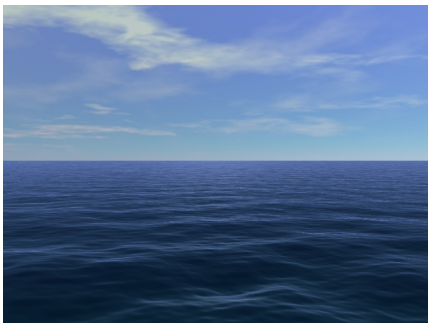


Figure 2. Simple antialiased scene

3.3 Coherence and data structure

As noted in [HW96], *spatial coherence* can be exploited since the evaluation $n_i(p)$ is bounded:

$$n_i(p') - L_i\|p - p'\| \leqslant n_i(p) \leqslant n_i(p') + L_i\|p - p'\|$$

where L_i is the Lipschitz bound and p' is a neighboring point. We can now estimate the value of $n_i(p)$, provided that $n_i(p')$ was evaluated before. This estimation is used to perform the prerequisite sorting stage of Algorithm 1, thus avoiding numerous function evaluations; as a counterpart, the global Lipschitz bound L_e may not be a maximum depending on the distance $||p-p'||$. In our implementation, we use estimates if $||p-p'||$ is lower

 ISBN 978-80-86943-98-5

than a maximum distance, which is larger than the distance between successive points along the ray.

Image coherence can also be exploited directly by the sphere tracing algorithm since, for a given frame, a sphere computed along a ray is guaranteed not to contain an intersection with the surface. Thus, if a ray intersects a sphere computed for a previous ray, then it may progress safely to the next intersection with this sphere. Otherwise, the sphere tracing is performed as usual (see Fig. 3a).

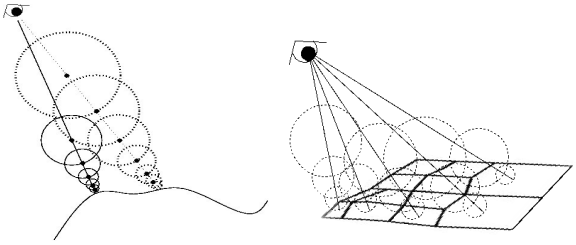


Figure 3. (a) Image coherence (b) Quadtree structure

It is noticeable that exploiting image coherence requires storing successive spheres along rays. Our data structure is a quadtree, aligned with the reference plane, which stores successive spheres along rays indexed by their intersection with the surface (see Fig. 3b). Any accelerating structure could be employed, but a quadtree seems a natural choice since fluid surfaces usually have regular shapes (especially ocean waves) which adapt well to axis-aligned partitioning. Following [HW96], image coherence is thus implemented by searching for previously computed spheres in a breadth-first manner, recursively investigating the neighbors of a ray at the same level in the quadtree. It is also worth noticing, as shown on Fig. 3b, that this data structure is well-adapted to antialiasing: subdivision simply depends on the distance to the camera.

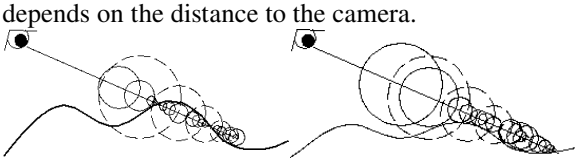


Figure 4. Starting from spheres traced at frame *t* (dotted), intersection at frame *t+1* can be closer (left) or further (right)

Our data structure is also useful to exploit *temporal coherence* between two successive frames if the camera does not move. In this case, spheres stored for a single ray at frame *t*, as well as the ray-surface intersection point p_t, can be re-used for the same ray at frame *t+1* (see Fig. 4):

1. Existing spheres from frame *t* are marked as *old*;
2. Sphere tracing is performed backwards (*i.e.* toward the camera) by re-computing the radius of existing spheres and marking them as *new*, until the maximal amplitude of the surface is reached. New spheres are also added to fill the gaps that may result;
3. The algorithm stops if the last intersection found is closer to the camera than p_t (Fig. 4, left);
4. Otherwise, sphere tracing is performed forwards starting from p_t (Fig. 4, right).

This strategy can also be adapted for image coherence: if a sphere intersecting a ray was marked as 'old', its radius is simply recomputed and marked as 'new'.

3. LIGHT-WATER INTERACTIONS AND COMPLEX PHENOMENA

4.1 Foam

We are interested in rendering foam which appears in ocean scenes far from the shore, also called passive foam, due to advection and turbulent diffusion of bubbles near the surface under determined conditions (*i.e.* wind speed greater than 13km/h). Fig. 5 shows a real picture of this phenomenon.

Figure 5. Real ocean scene with passive foam

We do not address in this paper active foam due to breaking waves, which has shorter lifetime and surface coverage, since our surface model does not represent this type of waves.

One approach to foam rendering in computer graphics makes use of particle systems [HW04] to simulate life and death of bubbles at the surface. Realistic results are obtained, but this solution can't be applied in the case of large ocean scenes. Another solution is to compute the amount of foam at a point on the surface according to different criteria: height variation between its neighbors [JG01], or amplitude of waves propagating around it [JBS03], which yields more foam in high-frequency areas. Unfortunately foam does not move in a realistic way between successive frames of an animation. Therefore, as in [PA01], we prefer another approach which considers different atmospheric conditions occurring in the process of foam apparition based on oceanographic results [MM86].

 ISBN 978-80-86943-98-5

4.1.1 Position

The amount of foam s covering the surface at a given point depends on wind speed U_{10} measured at 10 meters above the surface, and the difference between water temperature T_w and air temperature T_a:

$$s = 1.59 * 10^{-5} * U_{10}^{2.55} * e^{0.086*(T_w - T_a)} \qquad (3)$$

This empirical formulation is used in oceanographic research; but experiments show that it does not take into account the apparition of foam in regions more exposed to wind, especially on waves crests, nor in perturbed regions. Therefore, we propose to extend Eq. 3 in order to include the height of the considered point, and its local slope, thus obtaining the amount of foam s':

$$s' = s * \left(\frac{y - y_{min}}{y - y_{max}}\right)^n * \|\vec{N}\| \qquad (4)$$

where y_{min} (resp. y_{max}) is the minimal (resp. maximal) height and $\vec{N}$ is the (non normalized) normal vector at the considered point. Parameter n allows the user to control the spread of foam on waves crests; based on empirical experiments, our most visually convincing results were obtained with $n \in [6, 9]$. To avoid noticeable foam patterns repetitions at the surface, a noise function is added, filtered according to the distance to the viewer in the same way as the antialiasing technique described in section 3.2.

4.1.2 Radiance

Here again, an empirical formula for the radiance L_{foam} is suggested by oceanographers [Koe84]:

$$L_{foam} = s' * f_{ef} * R_e$$

where R_e is the radiance of pure white foam. The efficiency factor f_{ef} is originally defined as $0.4 +- 0.2$; to account for the foam's age, we modify it into $0.4 - 0.2.t.10^{-2}$ (truncated to 0 if negative) where t is the time parameter. Total radiance at the surface L_{tot} is then:

$$L_{tot} = L(0) + (1 - k_r) L_r(0)$$

$$L(0) = L_{sky} + L_{sun} \cos\theta_{sun} + L_{foam}$$

where $L(0)$ is the radiance on the surface computed by taking account the luminance of the sky L_{sky} (resp. sun L_{sun}), k_r is the Fresnel reflexion coefficient and $L_r(0)$ is the refracted radiance (see section 4.3).

To take into account the motion of foam during an animation, we use a texture in image space to store positions and ages of foam on the surface. For each new frame, a preprocessing step consists in incrementing age and displacing positions using formulas similar to Eq. 2. Then, for each pixel, the distances between its projected point on the surface and the neighboring points from the previous foam texture are considered:

1. if the distance is lower than a given threshold, the corresponding texel is used
2. if all the distances are greater than the threshold, a new texel is created in the foam texture using Eq. 4.

These distances are computed as soon as an intersection point between the corresponding ray and the ocean surface is found, thus the integration of this computation in our sphere tracing algorithm is straightforward.

Figure 6. Foam obtained with U_{10} = 13 km/h

Foam obtained with this method is shown on Fig. 6 for $U_{10} = 13$ *km/h*, $T_a = 20°C$ and $T_w = 12°C$.

4.2 Sprays

Oceanic spray can be seen as a gaseous exchange between the ocean and the atmosphere. It is mainly due to bubbles bursting at the surface, which release drops of diameter approximately one magnitude lower than bubbles'. Depending on their size, these drops can either go back in the water or evaporate. In the latter case, organic or gaseous molecules carried by the drops are left in the atmosphere, and thus form oceanic sprays.

In computer graphics, most existing methods simulate sprays using a particle system generated according to the surface height [JG01, HW04] or surface variations [JBS03]. But these methods do not take into account the different atmospheric conditions, and do not simulate the path of light going through sprays (as a participating medium). On the other hand, participating media in computer graphics is the subject of numerous papers (see [PCPS97] for a survey). A method is proposed in [PSB99] to characterize light attenuation and diffusion due to different atmospheric molecules in daylight. We propose to extend this work to the case of oceanic sprays.

Size distribution of sprays of radius r can be empirically computed as [MM86]:

$$\frac{dF(r)}{dr} = 1.373 U_{10}^{3.41} r^{-3} 10^{1.19} (1 + 0.0057 r^{1.05}) e^{-B^2} \qquad (5)$$

$$B = 0.380 \log(r) / 0.65$$

To take into account different atmospheric conditions, we modify this equation into:

 ISBN 978-80-86943-98-5

$$\frac{dF(r)'}{dr}=\frac{dF(r)}{dr}*s' \qquad (6)$$

where s' is the amount of foam computed from Eq. 4 at the orthogonal projection on the surface.

Furthermore, the vertical profile of sprays is described by [SF79]:

$$\frac{dF(r)}{dr}=\frac{dF(r,0)}{dr}e^{\frac{-z}{H_{max}}}$$

where $dF(r,0)$ is the size distribution at the ocean surface computed from Eq. 6, and H_{max} is the maximal height where sprays disappear. We now have a complete description of sprays distribution above the surface.

According to Beer-Lambert's law, the intensity of a light path going through a participating medium is diffused and attenuated. In the case of sprays, this attenuation is due to the sprays' optical thickness (AOD), defined by [PSB99]:

$$\tau_{aod}=\int_0^{z_{max}} k_e(z)dz \;\;,\;\; k_e=\int_{r_{min}}^{r_{max}} k(r)\frac{dF(r)}{dr}$$

$$k(r)=Q_e\pi r^2$$

where z_{max} is the maximal height above which sprays are negligible. Parameter k_e is called the extinction coefficient, and $k(r)$ is the efficient section of a spray of radius r. Parameter Q_e is called the attenuation efficiency (Q_e ~= 2 for oceanic sprays). The attenuation is then defined as:

$$L'(0)=L(0)e^{-\tau_{aod}}$$

Figure 7. Light attenuated by sprays

During sphere tracing, every time a new step is taken along a ray, the corresponding point is projected onto the surface and the amount of foam is computed, yielding to an attenuation coefficient defined for each sphere below z_{max} (since our algorithm makes use of spatial coherence as described in section 3.3, some spheres can be reused). The diffuse radiance for a given ray is obtained by integration of the diffuse radiance in all M spheres along this ray:

$$L(0)=L_{foam}+(L_{sky}+L_{sun}\cos\theta_{sun})\sum_{i=1}^{M} e^{-k_e(z)\Delta z}$$

where Δz is the vertical distance between two successive spheres.

The influence of sprays on the scene is shown on Fig. 7 for U_{10} = *18.5 km/h*, T_a = *20°C* and *Tw = 12°C,* and is noticeable only because light is attenuated.

4.3 Second order scattering

As in the previous section for the atmosphere, light is also greatly attenuated when going through water (a very small amount of light reaches 100m under the surface). This phenomenon is called second order (or sub-surface) scattering. Light diffusion in that case is mainly due to particles in suspension in the water volume, which can either be organic (phytoplancton) or formed of water molecules of different densities, and characterize the water's turbidity. Different water types (resp. A, B or C) correspond to different turbidities (resp. clear, turbid or very turbid).

Due to its complexity, second order scattering can only be simulated using hierarchical data structures: illumination volumes [IDT02], octrees [JB02] or horizontal planes [IDT03] were proposed. But these methods do not take particles in suspension into account. Another approach [PA01, CS04] consists in computing the diffuse radiance according to the type of water and its concentration of phytoplancton particles; we propose to simplify and adapt this idea to our data structure.

Following [PA01], the directional radiance $L(0,\theta,\varphi)$ in direction (θ,φ) re-emitted to the surface is defined as:

$$L(0,\theta,\varphi)=L(z,\theta,\varphi)e^{-cR}+L_{df}(z)$$
$$L_{df}(z)=L_{df}(0)(1-e^{(-c-K_d\cos\Theta)R}) \qquad (8)$$

where $L(z,\theta,\varphi)$ is the radiance at the bottom of the sea, $L_{df}(z)$ is the total diffuse radiance, c is the extinction coefficient of water, K_d is the diffusion coefficient, and $R = z/(\cos\theta)$.

$L_{df}(0)$ is the diffuse radiance at the surface:

$$L_{df}(0)=(0.33*b_b*E_d(0))/a\pi \qquad (9)$$

where b_b is the backscattering coefficient, inversely proportional to the phytoplancton concentration C, a is the absorption coefficient, and $E_d(0)$ represents the *downwelling irradiance* at the surface, which depends on atmospheric properties (interested readers should refer to [PA01] for more details).

We replace the term $E_d(0)$ in Eq. 9 by the diffuse radiance $L(0)$ computed by Eq. 7:

$$L_{df}(0)=(0.33*b_b*L(0))/a\pi$$

In that case, the ray is pointing downwards; the bottom of the sea is modeled as a set of trochoids, which allows to apply the sphere tracing algorithm to compute its intersection with a refracted ray. As in the previous section, a diffuse radiance is computed for each sphere by simplifying Eq. 8:

$$L_{df}(z)=L_{df}(0)(1-e^{(-c-K_d)z}) \qquad (10)$$

 ISBN 978-80-86943-98-5

with z the vertical distance between the surface and the center of the sphere. Finally, the total diffuse radiance for a refracted ray is obtained by integrating the diffuse radiance of all M spheres along the ray:

$$L_r(0)=(Z_{bottom})e^{-cZ_{bottom}}+\sum_{i=1}^{M} L_{df_i}(z)\Delta l \qquad (11)$$

where Δl is the distance along the ray and L_{dfi} is the diffuse radiance of each sphere computed by Eq. 10. In this case again, spatial coherence allows us to reuse diffuse radiance for previously computed spheres. Fig. 8 shows images obtained for clear and turbid waters and constant phytoplancton C = 1.0mg/m^3.

Figure 8. Clear and turbid water

5. RESULTS

5.1 Implementation

Our method is implemented as a plugin for Autodesk's Maya 6.5. In order to obtain more realistic results, a post-processing step is added for each frame to include glaring which is particularly frequent in ocean scenes, due to high contrasts between specular neighboring surfaces.

Figure 9. Glaring effect

Our implementation is an extension of a method proposed in [Tho01]: the most specular points on the surface are considered, and a "glare texture" is mapped on these points, whose size is determined according to the distance from the viewer. A minimal distance between two specular points is also introduced to avoid a concentration of glare points within a region, which would yield unrealistic results. An example is shown on Fig. 9, where all complex phenomena treated by our algorithm are also enabled (foam, sprays and light scattering).

5.2 Performances

Performances of our method were evaluated on different ocean surfaces defined by an increasing number of trochoids waves; the other parameters were constant. As expected, our method guarantees a significant speedup compared to brute-force ray-marching and unenhanced sphere tracing to compute intersections between a ray and the surface (see Table 1), increasing with the number of trochoids.

N	W. ray tracing	W. sphere tracing
30	45%	26%
40	49%	32%
50	50%	34%

Table 1. Speedup compared to brute-force ray-tracing (middle) and unenhanced sphere tracing (right) for 30, 40 and 50 trochoids (N)

Spatial coherence was evaluated on a single 640x480 image; temporal coherence was evaluated on a single 640x480 frame of an animation (see videos attached to the paper). Both comparisons were also significant (see Table 2). The total rendering time for a single image listed on the same table shows an interesting result: our method gives better results for a higher number of trochoids. This better convergence is due to our maximization approach, which tends to neglect most of the small amplitude waves with high frequencies.

N	SC	TC	Time
30	21%	24%	4:05mn
40	23%	30%	3:38mn
50	27%	39%	2:50mn

Table 2. Speedup obtained with spatial coherence (SC) and temporal coherence (TC)

Ray-surface intersections	62.5%
Sprays	18.3%
Second order scattering	12.4%
Glaring effects	3.7%
Foam	3.1%

Table 3. Repartition of different steps

 ISBN 978-80-86943-98-5

Table 3 shows the computations repartition between the different parts of the algorithm. Unsurprisingly, most of the time is spent to find intersections between rays and the surface, whereas foam is computed very quickly since most of this part is performed using textures.

6. CONCLUSION

This paper presents an efficient method to render realistic ocean scenes. A sphere-based data structure allows to accelerate ray-surface intersections computations by heavily relying on spatial and temporal coherence which are well-adapted to ocean scenes. To increase the realism of generated scenes, we also propose new formulations to integrate physically-based phenomena such as second order scattering, foam and sprays into our data structure, without significantly reducing performances. The limitations of our method are mainly due to the surface model, which can not represent breaking waves and physically-based foam and spray induced by this phenomenon; we are currently studying how to integrate this type of waves.

7. REFERENCES

[CGG01] Cieutat., J.M., Gonzato, J.C, Guitton, P., A new efficient wave modeling for maritime training simulator, SCCG, 2001.

[CS04] Cerezo, E., Seron, F.J, Rendering natural waters taking fluorescence into account, Comp. Animation and Virtual Worlds, 15(5), 2004.

[FR86] Fournier, A., Reeves, W.T, A simple model of ocean waves, SIGGRAPH, 1986.

[Har96] Hart, J.C, Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces, The Visual Computer, 12(10), 1996.

[HNC02] Hinsinger, D., Neyret, F., Cani, M.P, Interactive animation of ocean waves, SCA, 2002.

[HVT+06] Hu, Y., Velho, L., Tong, X., Guo, B., Shum, H., Realistic real-time rendering of ocean waves, Computer Animation and Virtual Worlds, 17(1), 2006.

[HW96] Hart, J.C, Worley, S.P., Hyper-rendering of hyper-textured surfaces, Implicit Surfaces, 1996.

[HW04] Holmberg, N., Wunsche, B.C, Efficient modeling and rendering of turbulence water over natural terrain, GRAPHITE, 2004.

[IDT02] Iwasaki, K., Dobashi, Y., Nishita, T., An efficient method for rendering underwater optical effect using graphics hardware, Vol. Graphics, 2002.

[IDT03] Iwasaki, K., Dobashi, Y., Nishita, T., A volume rendering approach for sea surfaces taking into account second order scattering using scattering maps, Vol. Graphics, 2003.

[Igl04] Iglesias, A., Computer graphics for water modeling and rendering: a survey, Future Generation Computer Systems, 20(8), 2004.

[JB02] Jensen, H.W., Buhler, J., A rapid hierarchical rendering technique for translucent material, SIGGRAPH, 2002.

[JBS03] S. Jeschke, H. Birkholz, H. Shumann. A procedural model for interactive animation of breaking ocean waves. Computer Graphics, 2003.

[JG01] Jensen, L.S., Goliàs, R., Deep-water animation and rendering, http://www.gamasutra.com, 2001.

[Koe84] Koepke, P., Effective reflectance of ocean withecaps, Appl. Optics, 23(18), 1984.

[Mit05] Mitchell, J.L., Real-time synthesis and rendering of ocean water, Technical report, ATI, 2005.

[MM86] Monahan, E., MacNiocaill, G., Oceanic Witecaps: Their Role in Air Sea Exchanges Processes, D. Reidel, 1986.

[PA01] Premoze, S., Ashikhmin, M., Rendering natural waters, Computer Graph. Forum, 20(4), 2001.

[PCPS97] Perez-Cazorla, F., Pueyo, X., Sillion, F., Global illumination techniques for the simulation of participating media, EG Workshop on Rendering, 1997.

[Pea86] Peachey, D.R., Modeling waves and surf, SIGGRAPH, 1986.

[PSB99] Preetham, A., Shirley, P., Smiths, B., A practical analytic model for daylight, SIGGRAPH, 1999.

[SF79] Schettle, E., Fenn, R., Models for the aerosols for the lower atmosphere and the effects of humidity variations on their optical properties, AFGL-TR-79-0214 676, Envir. Research Papers, 1979.

[TB87] Ts'o, P.Y., Barsky, B.A., Modeling and rendering waves: wave-tracing beta-splines and reflective and refractive texture mapping, ACM Trans. Graph., 6(3), 1987.

[Tes01] Tessendorf, J., Simulating ocean waters, SIGGRAPH Course Notes, 2002.

[TFK+03] Takahashi, T., Fujji, H., Kunimatsu, A., Hiwada, K., Saito, T., Tanaka, K., Ueki, H., Realistic animation of fluid with splash and foam, Eurographics, 2003.

[TG02] Thon, S., Ghazanfarpour, D., Ocean waves synthesis and animation using real world information, Computer and Graphics, 26(1), 2002.

[Tho01] Thon, S., Representation de l'eau en synthèse d'images, PhD thesis, Université de Limoges, 2001.

[TRS06] Thürey, N., Rüde, U., Stamminger, M., Animation of open water phenomena, SCA, 2006.

 ISBN 978-80-86943-98-5

Generation of Shadows in Scene Graph based VR

Bjoern Kuehl
University of Hamburg, Germany
9kuehl@informatik.uni-hamburg.de

Kristopher J. Blom
University of Hamburg, Germany
blom@informatik.uni-hamburg.de

Steffi Beckhaus
University of Hamburg, Germany
steffi.beckhaus@uni-hamburg.de

Abstract

In this paper, we present our experience implementing shadows in Scene Graph based Virtual Reality systems. Shadows are an important part of the human perception of both shape and depth and, yet, are a largely missing component in Virtual Environments. In this work, we investigate extending standard Scene Graphs to automatically produce shadowed scenes. This paper presents our experience embedding two popular real-time shadow methods, Shadow Mapping and Stencil Shadow Volumes, in a popular Scene Graph system, OpenGL Performer. Our experience has shown both ways in which they can be used and also a number of weakness in the current state of both Scene Graph systems and shadow methods. Based on our experiences, we present suggestions for the user desiring to include shadows in their Virtual Environment and highlight areas, where further development would benefit users significantly.

Keywords: Scene Graph, Virtual Reality, Stencil Shadow Volumes, Shadow Mapping

1 INTRODUCTION

A well known German figure of speech says: "There is shadow where there is light." This omnipresent effect of light occlusion makes "the shape and relative position of objects in such scenes more comprehensible" [11] and helps us to identify the distribution of light sources. However, in the realm of computer graphics, this figure of speech does not necessarily apply, because this familiar effect doesn't appear automatically, but must be imitated. The extra effort is useful, because the usual uniform illumination of computer graphics is easy to identify as artificial, amongst other things due to "flying" objects.

The perceptual cues provided by shadows are presumably even more important in the field of Virtual Reality (VR). VR focuses on both immersive environments and providing the illusion of depth; Shadows deliver important information about depth to the user. Given these foci, it is remarkable that shadows are unusual to find in the most VR environments. Perhaps, the biggest reason for this is the lack of integrated support in the graphics generation tools that VR systems typically use, Scene Graphs; Scene Graph(SG) systems used have only just begun to support shadows.

In recent years, two techniques have emerged as standards in the real-time computer graphics community for shadows generation. The first, used in the majority of cases, is **Shadow Mapping**. [11] Shadow Mapping uses a texture of the scene generated from the lights point of view, in combination with a special algorithm, to determine the portions of the scene that are not visible from the light and, therefore, in the shadow. The second shadow technique is **Stencil Shadow Volumes**. [2] This method marks objects as lying within the shadow if they are contained within a volume produced by extending the silhouette of a shadow caster with respect to a light.

In this paper we present the results of our investigation of integrating automatic shadow generation into Scene Graph based VR. In our work we have focused on SGI's OpenGL Performer^{TM} [6] - a popular SG, used in numerous VR systems - and the VR system, AVANGO. [10] We present here how shadows, both Shadow Mapping and Stencil Shadow Volumes, are integrated into AVANGO and describe the implementation of them briefly. However, a large portion of this work's importance lies rather in the experience collected during this exploration. We present much of that experience here, in the form of a number of suggestions for the user of such a system. In this discussion we also bring to the forefront many of the shortcomings of the shadow techniques for general usage and describe a number of points, where improvements to the techniques would be advantageous.

In the next sections, we present background information on Virtual Reality and Scene Graphs, followed by a small overview of the Shadow Mapping and Stencil Shadow Volume methods. In Section 4 we present details on the implementation of both shadow techniques in a SG-based VR system. In Section 5, we present the outcomes of this work, including giving advise how to use automatic generation of shadowing and discussion of the shortcomings of both the SG implementation and the shadow techniques. Section 6 discusses future di-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency - Science Press, Plzen, Czech Republic.

 ISBN 978-80-86943-98-5

rections for research we have identified that would improve shadow usage in the VR context.

2 VIRTUAL REALITY

In this section we provide some necessary background information on Virtual Reality, by contrasting it with other, more standard computer graphic areas. A special focus is placed on the generation of graphics in VR systems, performed by underlying Scene Graph (SG) systems. Finally, we give a brief overview of the related work in VR and similar areas on shadows.

VR, like computer games, builds highly upon the real-time computer graphics community. VR and computer games share many similarities; However, VR differentiates itself on a number of fronts. VR is highly focused on immersion, roughly the illusion of being present in the virtual environment. This is achieved through various means. For this feeling of presence, one factor is a high quality image. The "connectedness, continuity, consistency, and meaningfulness of the perceptual stimuli presented" [5] is denoted as "Pictorial Realism" and this realism can be increased by shadows.

The hardware normally used in VR can also increase this feeling of presence. Common components include surrounding (immersive) displays, such as large projection systems or Head Mounted Displays (HMDs), stereoscopic imagery. Another component that differentiates VR from computer games is that the user's position is usually tracked, having the user's physical movement affect the display of virtual environment. The resulting motion parallax effect "'provide important information about the depth of objects in the field." [5] Stereoscopic imagery and the paralax effect enabled with head tracking have significant effects on depth perception. These also have impacts on VR systems' structure. Large scale projection systems require multiple projections, implemented either with special multi-piped systems or implemented using distributed systems. Head-tracking leads most VR to use a "the world moves" metaphor instead of the common camera movement metaphor.

In contrast to computer games, VR exhibits two big disadvantages in regards to the modeled 3D environment. Among the VR community, it is uncommon to have dedicated modelers. Commonly, available models and models created by programmers and students with little or no experience must be used. A further factor is that many games, particularly those with shadows, take place within closed environments, such as buildings with small rooms. Conversely, the environments of VR are often very large and quite often consist of open spaces, both of which become critical factors when dealing with shadows.

2.1 Scene Graph Based Systems

A Scene Graph is used by most VR systems for organizing the scene. A Scene Graph (SG) is a directed, acyclic graph consisting of a hierarchical structure of nodes designed for the simplification of the display and management of graphical environments. SGs are typically tree structures, where the root node is a "scene node," on which the rest of the scene is hung. The nodes of the graph specify information about the scene and its properties, such as transformations, state properties, material properties and geometrical primitives. In this work we have focused specifically on OpenGL Performer™, referred to throughout simply as Performer. However, the work we present here is largely applicable, outside of implementation details, to the two other popular SGs in the VR community. A large majority of the VR systems use one of the three. OpenSceneGraph (OSG: http://www.openscenegraph.org/) is a large community project and has an API very similar to OpenGL Performer. OpenSceneGraph (OpenSG: http://opensg.vrsource.org/trac) is built on the same principles, but presents a highly different API.

To render the scene, the scene graph is traversed, rendering all visible objects. A **traverser** traverses the scene graph, typically originating from the scene node and proceeding to the leaf nodes, which contain the actual geometry. In between the root node and the leaf nodes are nodes that transform and change the scene layout. In Performer, the leaf nodes are of type **pfGeode** and contain one or more **pfGeoSets**, which hold the actual geometry primitives. To be able to position the geometry in the scene, the pfGeode is hung on a transform node, the **pfDCS** node. The final major component, **pfGeoState**, is responsible for influencing the graphical state of an object and is attached to individual pfGeoSet's.

The AVANGO VR system is built as a layer on top of Performer. AVANGO is an object oriented framework for creating distributed, interactive VR applications, with an interface that is similar to Open Inventor. [10] It supports using many different input devices and also supports various types of displays. As with most VR systems, AVANGO handles this by abstracting from the underlying devices. This abstraction is a boon to application writers, but also makes implementation of some advanced concepts difficult as access to necessary components may be denied.

The main components of AVANGO are the scene graph and an orthogonal dataflow *Field Container* concept. The state of a node or object is encapsulated in *Fields*. Fields can be connected to other fields to synchronize values between them. In this manner a dataflow graph is constructed, combining Performer derived nodes with pure AVANGO Field Containers.

 ISBN 978-80-86943-98-5

To get the best performance possible, all classes in AVANGO are coded in C++. To achieve increased flexibility and run-time coding capabilities, a binding to **scheme** is built in, including an interpreter for run-time command execution.

2.2 Related Work

Literature in regards to shadows in VR is very limited. The only paper directly related to our work is early work from Slater et al. [7] Their research was based on a custom Shadow Volume system and was focused on presence research. Unfortunately, the results were discouraging, but presumably are due to the very primitive technology they used.

The related areas of Augmented Reality and Mixed Reality have been more focused on shadows, particularly dealing with mixed real and virtual content casting shadows. Both shadow generating method were used to in the related literature. For example, in [3] Stencil Shadow Volumes are used, while in [9] Shadow Mapping is used to create the shadows. These works focus highly on the underlying graphical manipulations. Prior work could not be found, in which shadows were integrated into scene graph systems. However, all three of the major SG systems have created some implementation of Shadow Mapping.

The gaming community is the one area that uses both Shadow Mapping and Stencil Shadow Volumes. Examples are commercial game engines and also state-of-the-art graphic engines, for example the popular **Ogre** (http://www.ogre3d.org/). Unfortunately, these engines are difficult to adapted to usage in VR systems, most notably because of multi-threading issues.

3 SHADOW TECHNIQUES

In this section, we briefly present the theoretical foundations of the two main shadow techniques used in real-time graphics, Shadow Mapping and Stencil Shadow Volumes. This overview is purposely general and implementation details will be handled as required in Section 4. Where appropriate, references to more thorough resources are provided.

3.1 Shadow Mapping

The most popular shadow technique, used by a large number of today's games and simulation applications, is Shadow Mapping. [11] It is based on the idea that all geometry visible from the light must be lit, and all geometry that can not be seen from the light's point of view lies in the shadow. Shadow Mapping is capable of casting shadows on arbitrary surfaces and also supports self-shadowing and inter-object shadows.

For Shadow Mapping, a two-pass algorithm must be used. In the first pass, the scene is rendered from the light's point of view. The depth values are then written to an already prepared depth texture. To use this texture in the second pass, which is from the view of the camera, it must be projected onto the scene from the light's point of view by calculating appropriate texture coordinates.

Using the generated texture coordinates, created using OpenGL's automatic texture coordinate generation, the distance between the fragment and the light source can then be compared with depth stored in the r component of the corresponding texel(s) of the shadow map. If both values are equal, then the corresponding fragment is the same as the one seen from the camera's point of view. However, if the r component of the fragment's texture coordinate is greater than the corresponding texel, then the fragment can not be seen from the lights point of view. This indicates that there is another object between the fragment and the light source. Therefore, this fragment lies in a shadow.

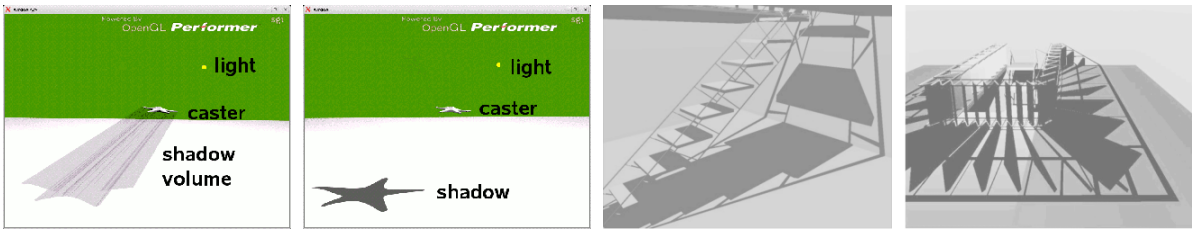
3.2 Stencil Shadow Volumes

The idea to use volumes to calculate shadows was introduced by F. Crow. [2] The volume that the shadow encloses is calculated for each light and object combination. It can then be used to determine what geometry is located inside the shadow. The silhouette of an object is formed from precisely the edges, for which one adjacent plane is facing the light and the other is facing away. To check if a plane is facing the light, one has to calculate the angle between the faces normal and the incoming light ray. If this angle is $> 90°$, the plane is facing toward the light source. To create a shadow volume, as seen in Figure 1(a), he extended the recovered silhouette by displacing copies of each silhouette's vertices in the direction opposite to the incoming light ray.

In order to use the later introduced GPU's inherit vertex processing power in calculating shadow volumes a new method had to be developed. Since vertex shaders are not capable of producing new vertices, objects have to be enhanced with new vertices, which can later be displaced by the vertex shader. To do that, all the object's edges are replaced by newly created quads. Two of the quad's vertices receive the normal of one adjacent face and the other two vertices the normal of other face. These quads are called "degenerate quads," since they don't contain any surface area at creation time. To create a shadow volume from the extended object, we use the vertex shader to displace all vertices that aren't facing the light source. Through the displacing process the degenerate quads are stretched so they get their own surface area. The created shadow volume consists of the newly emerged quads together with the displaced and the non-displaced triangles capping it.

To get shadows from the calculated shadow volumes, as seen in Figure 1, the counting capability of the stencil buffer is used to find these objects. [4] Heidmann

 ISBN 978-80-86943-98-5



(a) A scene with a visible shadow volume (b) the shadow is visible, where the volume intersects geometry (c) stencil shadows in a real environment (d) stencil shadows in a real environment

Figure 1: Stencil Shadow Volumes

followed Crow's method for determining which objects are shadowed, by counting the order of the objects and volumes surfaces, today known as **z-Pass** or depth pass. It is a method, in which the front and back faces are successively rendered. If a fragment of the front faces can be drawn, the associated stencil value is incremented. Afterwards, the back faces are rendered and, for every fragment that isn't covered by another object, the associated stencil value is decremented. Fragments with an associated stencil value > 0 after all faces are rendered indicate that there is an object within the volume; Therefore, it lies in the shadow. After completing this step for all fragments, the stencil buffer contains values that divide the scene into lit and shadowed regions.

Unfortunately, z-Pass inverts the lit and shadowed regions if the camera is located inside the shadow volume. To avoid this problem, the order of operations can be reversed, known as **Carmack's reverse** [1] or simply as **z-Fail**. Using this method, the stencil buffer must be incremented, if it was not possible to draw the volume's back face, and decremented, if the front face cannot be rendered. In contrast to z-Pass, it is necessary to cap the volume, making z-Fail a bit slower. The results are almost the same, but enables entering a shadow volume, without producing inverted shadows.

There are two ways in which the stencil buffer's mask can be used to render shadows. One can use either "modulative shadows" or "additive light masking." Using modulative shadows, we overlay the scene with a quad, colored with a semi-transparent shadow color. Doing this exclusively in the regions masked in the stencil buffer, the scene's color is darkened inside the shadowed region. This method is comparatively fast and easy to use, but can produce errors within the shadow if there is specular highlighting inside the shadowed region. The "additive light masking" approach produces more realistic shadows and handles highlights the correct way, but it is noticeably slower, since the image must be drawn twice. In the first pass, the scene is rendered only with weak ambient light. After determining the shadowed regions, the scene is rendered again to the regions outside the shadowed regions with fully enabled lighting.

4 IMPLEMENTATION

The goal of our work was to make standard shadow methods available in the scene graph based VR system AVANGO, such that they were automatically created throughout the entire scene. This goal was based on an outsider's view that the shadow methods were, at this point "plug 'n play," and it should be just a simple matter of adapting the processes to work with SG based systems. In this section we present the implementation details of how both Shadow Mapping and Stencil Shadow Volumes can be implemented into SG based VR systems, specifically OpenGL Performer and AVANGO. In the following section we discuss the results of our implementations.

4.1 Shadow Mapping

Integrating Shadow Mapping into a VR system isn't necessarily easy, due to it being implemented as a two pass algorithm. While the first pass is only necessary when the light sources moves, VR systems use a "the world moves" metaphor instead of moving the camera, which means the light sources move every frame.

In Performer multi-pass algorithms can be implemented using two or more **channels**, conceptually a view of the scene that is part of the underlying "pipe." In many VR systems, including AVANGO, the underlying windowing, Performer's pipes/channels, is hidden from the user. This makes it difficult to create a second perspective needed for the Shadow Mapping method, as the underlying VR system must be modified.

Fortunately, it turned out that creating our own Shadow Mapping implementation wasn't necessary. Our initial investigations on integrating Shadow Mapping lead us to look at the Performer's own implementation, which was supported officially only under Windows and AVANGO functions only under Linux, to see how it was dealt with there. This investigation showed that the Performer method, supported through the class **pfLightSource**, did indeed function under Linux as well. Instead of reimplementing Shadow Mapping ourselves, we proceeded with the version available and describe here how it functions and how it is then integrated into AVANGO.

These light sources are SG nodes, which means that the position, from where the scene is lit, is determined

 ISBN 978-80-86943-98-5

by placing them in the SG. Because Shadow Mapping doesn't support omnidirectional shadows, it is important to specify the light's direction. The field-of-view (*fov*) of the light source is an important factor affecting the quality of the shadow; A smaller *fov* can better take advantage of the texture resolution, which results in smoother borders of the shadow. For the precision of the shadows, the two clipping planes must lie as close together as possible. In AVANGO, these configurations can be changed at run-time, since all attributes of the class fpLightSource are represented as fields. The following code snippet shows the sequence of scheme scripting commands required to use Performer's Shadow Mapping in AVANGO:

```
(define newLight (make-instance-by-name "fpLightSource"))
(fp-set-value newLight 'Name "newLight")
(fp-set-value newLight 'Position (make-vec4 0 0 0 1))
(fp-set-value newLight 'SpotDir (make-vec3 0 0 -1))
(fp-set-value newLight 'SpotCone (make-vec2 0 180))

(fp-set-value newLight 'ShadowEnable 1)
// resolution of the shadow texture:
(fp-set-value newLight 'ShadowSize 512)
(fp-set-value newLight 'ShadowFrustum
          (make-vec4 -0.5 0.5 -0.5 0.5))
(fp-set-value newLight 'ShadowNearFar (make-vec2 0.01 100))
```

4.2 Stencil Shadow Volumes

As Stencil Shadow Volumes are found neither in Performer nor AVANGO, we describe here all of the implementation details from our implementation. We extended AVANGO with a new node, "fpStencilShadow," which is directly inherited from the node fpDCS. The realization of this extension, as a transform node, makes it capable of being positioned within the scene graph and also has the advantage of collecting all the required objects for this method under one node. The task of this node class is to extend loaded objects by degenerate quads and manipulate the way the underlying geometry is drawn.

To extend an object by new quads, it must be committed to the extension's most important field, "Caster." This field holds a node or complete sub-tree, allowing the inclusion of any geometry set consisting of triangles or triangle strips. When the field update mechanism finds that the Caster field has received a new object, a function is called that creates a custom recursive pfuTraverser, which traverses the object of the field, searching for geometry nodes. When a geometry node is found, the contained GeoSets are extracted to create the new quads from them. Since GeoSets can contain only one primitive type, the new quads are put into a newly created GeoSet.

To create the shadow volume, the newly created GeoSet and a copy of the original one are placed as children of two newly created geometry nodes. This is done, so that the front and back faces can obtain different Draw-Traverser functions, describing the different behaviour for passing or not passing the depth test. These two nodes are then attached to the instance of the fpStencilShadow class.

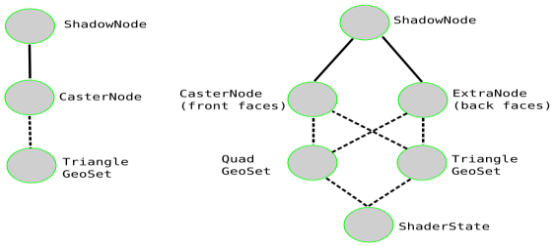


Figure 2: An object consisting of one triangle-geoset is hung under the ShadowNode. After modification performed by the traverser, another geode is created referencing both the triangle and the quad geosets.

The resulting Scene Graphs can be seen in Figure 2. In order to be able to displace the vertices of the new GeoSets later, all GeoSets have a special GeoState that is comprised a shader program that displaces vertices, dependant on the light's position. Further state requires are discussed later.

In Performer, it is possible to use OpenGL commands directly, so we can change the systems state. Unfortunately, changing the systems state this way is critical, because the changes are hidden from Performer. This means that Performer acts on the assumption that the state isn't touched, so completely unwanted effects can occur. That is why we have to save the OpenGL state on our own, so we can undo the changes later.

To be able to undo these changes, the first command stores the current state. In the Post-Draw-Function, this saved state can be loaded again. In order to perform the depth test without changing the depth entries, the depth test is enabled and the associating mask is "false." The Pre-Draw-Function shown below is designed for the front faces, so the back faces are culled away. Now, the stencil test can be enabled and configured, such that it is incremented when it is possible to render the current fragment. Using z-Pass, the Pre-Draw-Traverser for the front faces follows:

```
int fpStencilShadow::pfdPreDraw1(pfTraverser *trav, void *data)
{
        glPushAttrib( GL_ALL_ATTRIB_BITS);
        glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
        glEnable( GL_DEPTH_TEST);
        glDepthMask(GL_FALSE);
        glEnable( GL_CULL_FACE);
        glCullFace( GL_BACK);
        glEnable(GL_STENCIL_TEST);
        glStencilFunc(  GL_ALWAYS, 0, 0xffffffff);
        glStencilOp( GL_KEEP, GL_KEEP, GL_INCR_WRAP);
   return PFTRAV_CONT;
}
```

After attaching the shadow volumes to the scene graph, they are rendered, with the color buffer disabled. This is done in order to create a mask in the stencil buffer that separates lit regions from shadowed ones. As mentioned previously, there are two ways of using this mask for creating real shadows. We have implemented "modulative shadows," in order to avoid the multi-pass issues discussed in Section 4.1. Therefore, a node containing a quad is held in the field **ShadowQuad**. The ShadowQuad functions as a semi-transparent film, through which the whole scene can be

 ISBN 978-80-86943-98-5

(a) 32x32 (b) 128x128 (c) 256x256 (d) 1024x1024

Figure 3: Depiction of a typical outdoor scene, a marketplace in Germany with different sizes of depth texture.

seen, darkening the shadowed regions. For this function to behave correctly, it is necessary that the film is always positioned between the camera and the scene. For this function to behave correctly, it is necessary that the film is always positioned between the camera and the scene. For that reason, a transformation node, above the geometry node, positions the film in front of the camera.

Furthermore, if the field that contains the ShadowQuad is modified, the field update mechanism calls a function that creates a new traverser. It then traverses the geometry subtree searching for geometry nodes to attach the Draw-Traverser functions to them. These functions configure the drawing of the ShadowQuad, such that it is drawn dependent on the values in the stencil buffer. In this manner, the film is laid over the scene only on the shadowed regions. In order to avoid having the shadow quad simply covering objects behind it, the Draw-Traverser activates the blending mode, darkening the shadow regions by blending its color with a black color whose alpha value is 0.5. The code of the Draw-Traverser function is as follows:

```
int fpStencilShadow::pfdPreDraw3(pfTraverser *trav, void *data)
{
        glPushAttrib( GL_ALL_ATTRIB_BITS);
        glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
        glEnable( GL_STENCIL_TEST);
        glStencilFunc(GL_NOTEQUAL, 0,  0xffffffff);
        glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE);
        glColor4f(0.0f, 0.0f, 0.0f, 0.5f);
        glEnable(GL_BLEND);
        glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        return PFTRAV_CONT;
}
```

5 RESULTS

In this section, we discusses our experience of using the implemented shadow methods. We also provide the user with much of the information that they will require in order to successfully introduce shadows into their environment. We have divided this into three parts, handling of the two method individually and a short statement on using both methods simultaneously.

Shadow Mapping The Shadow Mapping routine works much as advertised, working with arbitrary surfaces and performing self-shadowing. As seen in the code in Section 4.1, it is easy to enable in AVANGO. It is, however, not a "plug 'n play" method. As will be discussed in the following paragraphs, there are numerous points to tweak to the individual scene. One of the main points that requires careful consideration from the user is the texture resolution vs. the area to which it is applied. The second area of importance relates to the depth comparison and its inherit inaccuracy. Luckily, the AVANGO implementation assists the user highly in this endeavour, as they can modify all relevant values at runtime using the scheme interface or a GUI interface.

Depending on the resolution of the shadow texture, the shadow is, more or less, blocky. Although it is possible to assign an arbitrary value for the resolution of the shadow texture, it must be a square of a power of 2 to be accepted as a valid value. To get a correct texture mapping, the texture resolution can be no larger than the window size and the TFT monitor used had a maximum resolution of 1280x1024, so already a texture resolution of 1024x1024 could only be used in full screen mode. Naturally, larger textures take longer to generate and can cause texture swapping problems, so always selecting a large texture may not be appropriate.

As one can see in Figure 3, any resolution $<=$ 128x128 results in such a blocky shadow that it is better to leave the scene shadowless than use a shadow that can be identified as artificial on the first view. We recommend the use of the maximum resolution available, or at least 512x512 as a resolution for the shadow texture. The results are still blocky, but Performer tries to soften the edges, so it is tolerable.

The other half of the resolution problem is dependent on the chosen field-of-view and, in practice highly, influences the "blockyness" of the shadows. The smaller the field-of-view is, the better the resolution is utilized. In VR, where it is common to simulate a world size of square kilometres, it is not recommend to use a shadow field-of-view including the whole scene like it would be used for a sun to casts shadows. Instead, local light sources lighting need to be used, shadowing only a small area with each.

Additionally, it is possible that the depth value of a fragment in the depth texture differs from the one stored in the frame buffer for the same object, since the depth values of the frame buffer and the ones of the shadow texture descend from different rasters. This produces artefacts and incorrect self-shadowing when they are compared and are detected as different. The steeper the slope of the surface in depth and the lower

 ISBN 978-80-86943-98-5

(a) polygon offset is: too high (b) polygon offset is: too low (c) polygon offset is: just right (d) both techniques in combination

Figure 4: Generating and tweaking shadows using the implementation.

the resolution it is, the more such errors appear. The special values PFLS_SHADOW_DISPLACE_SCALE and PFLS_SHADOW_DISPLACE_OFFSET reduce such mistakes by adding a small offset, which is dependent on the surface's depth slope, to the sampled depth value. Usual values are 1.0 for the displace scale and 0.001 for the displace offset, but they are scene dependent. Customizing them for the current scene must be done manually. The effect of an incorrect offset, either too great or too small, can be seen in Figure 4. An important thing to mention is that it often occurs that when a good value for an specific object in the scene is found, it will result in improper results for other objects. It may be the case that there is no value, with which all objects are properly shadowed!

Stencil Shadow Volumes The Stencil Shadow Volumes implementation integrated well in AVANGO and seemed to be a solution that would be highly effective for our use. The implementation process was made more complex due to Performer's handling of state. Objects in Performer that do not possess their own GeoState inherit the current GeoState. This is done to reduce OpenGL context switches, but leads to problems if extra care is not taken in the shadow programming. That is why the GeoSet of the shadow quad also gets an empty GeoState to avoid inheriting the current one. With this implementation issue taken care of, we had expected the user to get their "plug 'n play" shadows, if only for limited objects due to the processing load of the method.

While implementing the Stencil Shadow Volumes, we discovered fully unexpected problems with regard to the shadow casting objects. Although it was known that such objects must be closed, 2-manifold, and consisting only of triangles (or triangle-strips), the problems these limitations brought up were not foreseen.

Most of the objects available on the internet are not constructed carefully enough to be usable for shadow volume generation, because the modelers didn't create them for this purpose. Similarly, the models created by students and computer scientists are often inadequate. If the model is only displayed, it seems to be all right, but if further processing of the model is performed, it causes problems. Shared vertices of neighboring surfaces that have differing positions and are bordered by more than two surfaces could result in holes. This leads to incorrect results, not only for shadow volume calculation, but also for techniques like Non-Photorealistic Rendering. A single incorrect vertex is enough to get a completely wrong shadow volume, which results in visible parts of the shadow volume.

This problem is unfortunately compounded by the SG systems and, in particular, Performer. When loading objects, the Performer loader tries to optimize the model for faster processing, converting it into its own native format. This leads to a re-triangulation of the object, which, in turn, isn't implemented carefully enough to create shadow volumes from this. Many times, after this "optimization," incorrect planes are included in the object or other planes are missing, so there are holes in the object. Mostly, these faults are not visible on the object itself, but the models can no longer be used for shadow generation. To avoid such problems, it is possible to convert the object to the Performer format in advance by using the program "pfConv." This application can be configured such that the triangulation remains untouched. If the converted object is loaded by the shadow-application, the loader will never re-triangulate it again.

Combined Usage Both the Stencil Shadow Volume and Shadow Mapping techniques and implementations have strengths and weaknesses. The quality of pictures strongly depends on the environment they are used on.

Most often, Shadow Mapping's implementation is faster than Stencil Shadow Volumes, since most of the calculations can be done on the specialized GPU of modern graphic cards. The dependency on textures and their limited resolution are two limitations on Shadow Mappings ability to produce beautiful shadows. On the other hand, it is not dependent on the scene's complexity as shadow volumes are, so it can be used as a more general solution, particularly for complex scenes. The Shadow Mapping implementation is also much more robust in terms of handling arbitrary objects, but requires fine adjustments with polygon offset over all components. Adjusting the polygon offset of the scene can be performed for static scene; However, for moving objects, changing position and form, the use Sten-

 ISBN 978-80-86943-98-5

cil Shadow Volumes do not require adjusting it to the current situation and scene, making potentially better suited for this usage.

Due to the completely different concepts used, it is possible to use both shadow methods in combination. The only thing that should be observed is the use of different light sources for each shadow method to avoid having the resulting shadows overlap. An example of the combined usage can be seen in Figure 4(d), where the brighter shadow is produced by Shadow Mapping. The darker coloring of the Stencil Shadow in the figure is purposely done to differentiate the two shadows.

6 FUTURE WORK

Our experiences have shown that the biggest problem with Stencil Shadow Volumes in the SG are objects with wrong surfaces or holes in them. To avoid these problems completely, methods to automatically check and repair the objects would be optimal. Such tools embedded in the modeling program would be of great help for modelers, particularly those amateur modelers VR often uses. Additionally, a model loader that is capable of reducing automatically the complexity of the GeoSets to accelerate the volume calculations would be desirable. In many Stencil Shadow Volume applications there is, next to the original object, a reduced version; The original is displayed in the scene and the reduced version is used for calculating the shadow volume.

Future investigations of interest would be the implementation of advanced Shadow Mapping approaches. There are many proposals that avoid or reduce the dependency on the texture resolution, for example the "Perspective Shadow Mapping." [8] Unfortunately, to use them in our setup would require customizing the available pfLightSource, which is not available in source code. Instead, like the fpStencilShadow extension, a new node or object must be created. This would require a multi-pass rendering algorithm, which needs to access the underlying channel to be implemented. Dependent on the VR system being used, this could be difficult to implement, as is the case with AVANGO.

7 CONCLUSION

In this paper we have presented how two popular shadow techniques, Shadow Mapping and Stencil Shadow Volumes, can be implemented in a Scene Graph based VR system, in pursuit of a system for automatic generation of shadows. We presented how we implemented these in the OpenGL Performer Scene Graph and the AVANGO VR system, using methods viable for other systems. Based on our experience, we have discussed both the implementation and usage of the systems. As a portion of this discussion, we have presented advice to the potential user of such a system on the different aspects that the user must be aware of, such as model issues with Stencil Shadow Volumes and tweaking Shadow Mapping parameters for typical VR environments. Finally, we have identified a number of areas of further research and development, which could help deliver the goal of truly automatic generation of shadows in SG based VR.

REFERENCES

[1] John Carmack. John carmack on shadow volumes. *http://developer.nvidia.com/object/robust-shadow-volumes.html*, 2000.

[2] Franklin Crow. Shadow algorithms for computer graphics. *Proceedings of SIGGRAPH 77*, 11(2):242–248, 1977.

[3] Michael Haller, Stephan Drab, and Werner Hartmann. A real-time shadow approach for an augmented reality application using shadow volumes. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 56–65, 2003.

[4] Tim Heidmann. Real shadows real time. *IRIS Universe*, 18:28–31, 1991.

[5] Wallace Sadowski and Kay Stanney. *Handbook of Virtual Environments: Design, Implementation, and Applications*, chapter Presence in Virtual Environments, pages 791–796. Lawrence Erlbaum Associates Inc,US, 2002.

[6] SGI. OpenGL Performer Programmer's Guide. *http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=0650&db=bks&cmd=toc&pth=/SGI_Developer/Perf_PG/*, 2004.

[7] M. Slater, M. Usoh, and Y. Chrysanthou. The influence of dynamic shadows on presence in immersive virtual environments. In *citeseer.ist.psu.edu/slater95influence.html*, 1995.

[8] Marc Stamminger and George Drettakis. Perspective shadow maps. In John Hughes, editor, *Proceedings of ACM SIGGRAPH 2002*. ACM Press/ACM SIGGRAPH, July 2002.

[9] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior augmented reality registration by integrating landmark tracking and magnetic tracking. *Computer Graphics*, 30:429–438, 1996.

[10] Henrik Tramberend. *Avocado: A Distributed Virtual Environment Framework*. PhD thesis, Universität Bielefeld, 2003.

[11] Lance Williams. Casting curved shadows on curved surfaces. *Proceedings of SIGGRAPH 78*, 5:270–274, 1978.

 ISBN 978-80-86943-98-5

Visual Attention for Significantly Influencing the Perception of Virtual Environments

Jasminka Hasic*
Sarajevo School of Science and Technology
Bosnia and Herzegovina

Alan Chalmers†
University of Bristol
United Kingdom

Abstract

The Human Visual System (HVS) is a key part of the rendering pipeline. The human eye is only capable of sensing image detail in a 2° foveal region, relying on rapid eye movements, or saccades, to jump between points of interest. These points of interest are prioritised based on the saliency of the objects in the scene or the task the user is performing. These "glimpses" of a scene are then assembled by the HVS into a coherent, but inevitably imperfect, visual perception of the environment. In this process, much detail, which the HVS deems unimportant, may literally go unnoticed.

In this paper we use knowledge of the HVS to influence what our attention is attracted to in computer graphics imagery, and thus what we actually perceive in those images. We influence the affinity of subjects towards an object based on the complexity of the context that object is put into. The images are rendered using the Radiance lighting simulation system. In this way, we are able to significantly influence users' preferences in an e-commerce application. Detailed psychophysical studies are used to validate our approach.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism-Colour, shading, shadowing, and texture—;

Keywords: Visual perception, attention, saliency, e-commerce

1 Introduction

Computer graphics imagery is increasingly playing a key role in e-commerce applications. High quality rendered images and virtual/augmented reality environments are now regularly used to help buyers select the product of their choice. One such example is choosing furniture [ARIS 2003]. This computer graphics imagery is ultimately seen by the Human Visual System (HVS) and while the HVS is good, it is not perfect, and is very much influenced by the scene being considered [James 1890]. The HVS does not process an image sequentially in a raster-like fashion, but rather our eyes rapidly "jump" through rapid eye movements, known as saccades, between features of interest in the scene. Our visual attention is a coordinated action involving conscious and unconscious processes in the brain. This coordinated action enables us to, rapidly and efficiently, find and focus on relevant information within a potentially cluttered scene. While the quality of the rendered image of a product itself is of course important, the choice of context also plays a significant role in capturing the HVS's attention.

In this paper, we investigate the perception of objects based on the context they are put in. Our target application is the sale of cars using both static images and dynamic web-based environments. The subjects considered were young adults, in particular students of the University of Sarajevo and Sarajevo School of Science and Technology. A detailed study, involving 114 subjects is carried out to determine their strong preferences for choice of car colour. We understand that car colour preferences vary across different cultures and age groups. Throughout the project, we kept the variance in cultural characteristics and age of subjects at a minimum.

We then demonstrate that this preference can be significantly influenced by the choice of context and other perceptual cues in a rendered scene, in particular a dynamic background object.

The rest of the paper is structured as follows. Section 2 details previous work in computer graphics which has exploited the human visual system. Section 3 describes the experiments that were conducted. The results of these experiments are given in section 4, and finally, in section 5, conclusions are drawn and avenues for future work presented.

2 Background

In 1890, James described the two general visual attention processes, termed *bottom-up* and *top-down*, which determine where humans locate their visual attention [James 1890]. The bottom-up process is automatic, without volitional control, and is purely stimulus driven. Our eyes are automatically attracted to, for example, any movement, especially in our peripheral vision, certain colours, the size, shape, brightness, edges and orientation of objects. This is evolutionary; the movement may be a predator lurking in the bushes, and our ancestors needed to be able to easily discern at a distance the red ripe fruit amongst the green of the trees. Top-down processes, on the other hand, are task dependent causing the HVS to focus on only those parts of a scene which are necessary for the user's current task, for example looking for street signs, or targets in a game [Yarbus 1967]. When attention calls for concentrating on an object or task, the viewer will often fail to perceive an unexpected object, even if it appears at fixation. This is known as Inattentional Blindness [Mack and Rock 1998]

Models of the human visual system have been used in

*e-mail: jh@ssst.edu.ba
†e-mail: alan@compsci.bristol.ac.uk

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency Science Press, Plzen, Czech Republic.

 ISBN 978-80-86943-98-5

computer graphics to investigate the perceptual quality of images [Daly 1993; Myszkowski 1998; Bolin and Meyer 1998], improve the quality of rendered images [Ferwerda et al. 1996; McNamara et al. 2000; Myszkowski et al. 2001; Pattanaik et al. 1998; Ramasubramanian et al. 1999], or to reduce the complexity of models without any perceptual loss of quality, such as [Luebke and Hallen 2001; Reddy 1997; Watson et al. 2001]. A good overview of perceptual rendering can be found in [O'Sullivan et al. 2004]. More recently, researchers have used detailed models of the bottom-up (known as Saliency Maps [Itti and Koch 2000]) and top-down (known as Task Maps [Cater et al. 2003]) visual attention processes to significantly reduce the computation of high-fidelity global illumination calculations [Yee et al. 2001; Haber et al. 2001; Cater et al. 2003; Sundstedt et al. 2004].

In this paper we use the saliency of objects to direct visual attention to different parts of a scene and thereby influence the manner in which a viewer perceives the scene.

3 Model and the Experiment

3.1 Model and Preference Defining Property

The environment chosen for our experiments consists of a car modeled in Maya and rendered using the Radiance lighting simulation system, as shown in Figure 1.

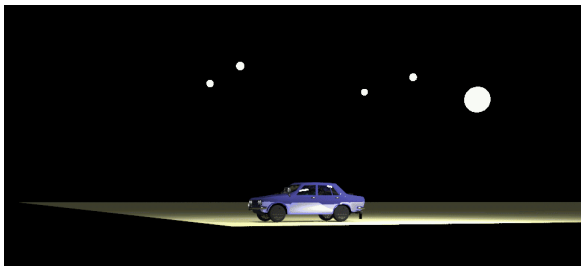


Figure 1: A radiance 3D scene

In order to carry out the experiments, we first need to establish a property of the car for which we have significant evidence that when altered will change the subjects' preference of the whole car. In [Hasic and Chalmers 2006] we showed that given the same car, if we just vary the car colour, the subjects, in general, prefer a blue car over the purple one. In this work 32 (16 male and 16 female) subjects were given the task to rank the chosen 6 car colours according to which car they would prefer to buy (1 - the most preferred car, 6 - the least preferred car). The background used in this experiment was neutral (medium grey). All car colours, Figure 2, were presented on one screen at once.

The table below summarizes statistics obtained for both male and female subjects. The number next to the car colour represents the average ranking of the particular car colour.

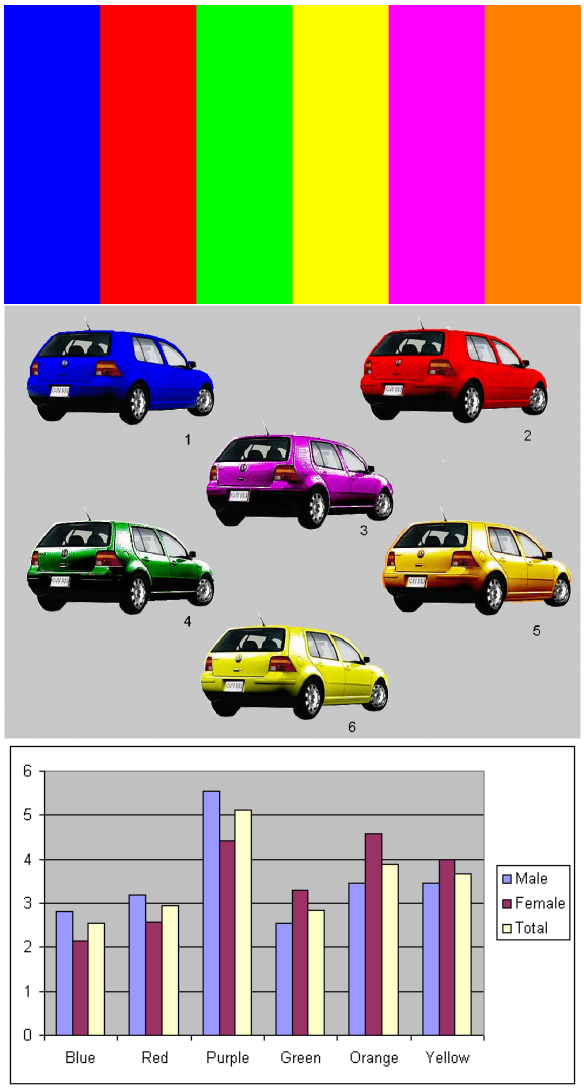


Figure 2: The different car colours shown and the subjects' preference

Female		Male		Total	
Colour	**Rank**	**Colour**	**Rank**	**Colour**	**Rank**
Blue	2.14	Green	2.55	Blue	2.56
Red	2.57	Blue	2.82	Green	2.83
Green	3.29	Red	3.18	Red	2.94
Yellow	4.00	Yellow	3.45	Yellow	3.67
Purple	4.43	Orange	3.45	Orange	3.89
Orange	4.57	Purple	5.55	Purple	5.11

This experiment established the blue car colour as most preferred and the purple car colour as least preferred one. To verify these results, another set of 16 subjects was asked to rank just those two car colours. The following preferences were obtained (see Figure 3):

Preferred Blue	Preferred Purple
86%	14%

The results demonstrate a clear difference in preference between those two car colours for our demographic. The blue and the purple car colours were used to represent preferred and non-preferred objects in the experiments conducted for

 ISBN 978-80-86943-98-5

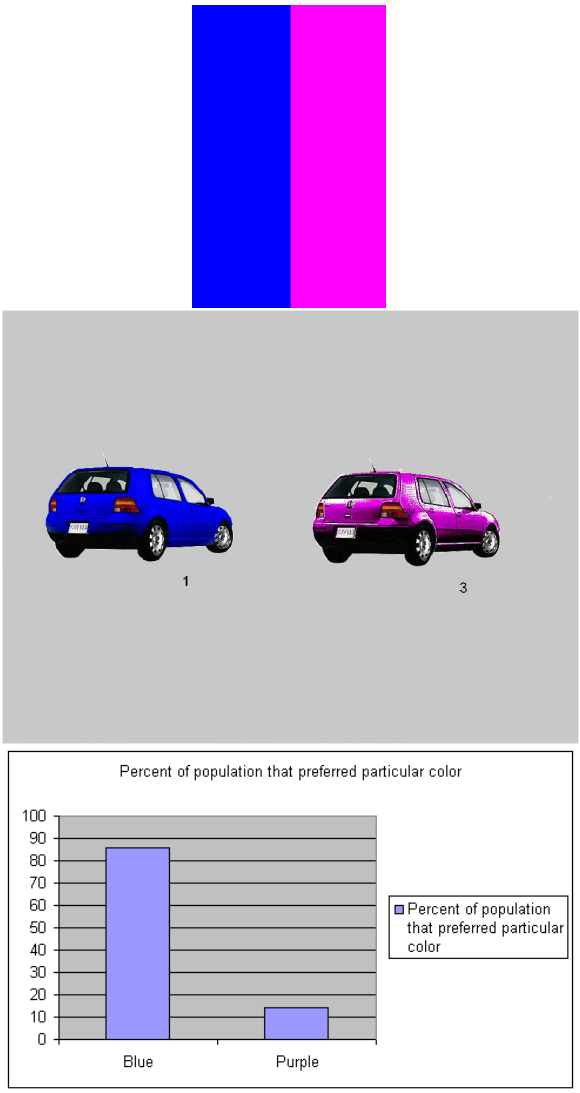


Figure 3: The best and the worst ranked car colour

this paper.

3.2 Experimental Setup

Two identical displays, with equivalent settings, placed next to each other were used in our experiments. 62 males students with normal, or corrected to normal vision, were the subjects for the experiments that followed. A pilot study was performed with 10 participants to determine the time required to comfortably complete the given task. Based on this pilot study the subject's response time was limited to 10 seconds. Most subjects responded well before this time limit. Due to the low number of female subjects in the overall student population used for this experiment, we chose to exclude answers from female students.

Each subject was seated in front of the two displays and had the same view of the both displays. On the two displays, a pair of images was presented and each subject was asked one question (not all subjects were asked the same question). Only one subject was present in the room where the experiment was conducted. Upon completion of the experiment, he was asked to keep the details of the experiment confidential and not to share any information about the experiment with other subjects since their experimental performance could be influenced by such information. The position of images in the two displays, within each pair, on the two displays was randomized to avoid bias.

4 Results

4.1 Comparison of objects on identical backgrounds

The results achieved in [Hasic and Chalmers 2006] about car preference based on the colour used 2D imagery. Since in this paper we are using a different car model and a global illumination 3D environment we had to verify that these earlier results would be consistent under the new conditions. The previous 2D imagery experiment tested the preference of car based on the car colour if the cars were placed on a neutral background (medium grey). In the first part of the experiment we verified that subjects would still prefer blue cars over purple ones if presented to them in the same context (on the same background). The two images were presented concurrently to one subject at a time and the subject was asked to answer the following question:

"Which car do you prefer?"

4.1.1 Car preference on neutral background

Each of the 10 subjects in this group was shown the pair of images in figure 4.

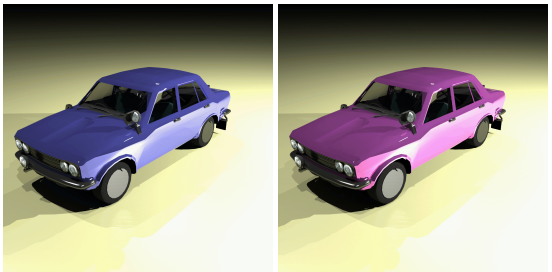


Figure 4: Car on neutral background

The following preferences were obtained from this group of subjects:

Preferred Blue	Preferred Purple
70%	30%

As we can see, this experiment confirmed that subjects do indeed prefer the blue new car model over the purple one on the neutral background.

4.1.2 Car preference on colourful background

The next group of 9 subjects was shown the pair of images in figure 5. This time the two cars were placed in a showroom-like setting. On the showroom walls colourful pictures were hung.

 ISBN 978-80-86943-98-5

Figure 5: Car on complex background

The results for this experiment were:

Preferred Blue	Preferred Purple
78%	22%

Even though the background of the two cars was changed from neutral to a more complex one, the subjects still highly preferred the blue car over the purple one.

4.1.3 Car preference on dynamic background

A final group of 9 subjects, in this part of the experiment, was shown the pair of images in figure 7. The background of these two cars was the same showroom-like background from the previous experiment, but with a dynamic addition. An animated button was added to the two pictures that were hanging in the show room. The face of the button had a label which read "CLICK HERE". As time went by, the face of the button opened in a window-like fashion and a new message was revealed reading "TO FIND OUT MORE" (see figure 6).

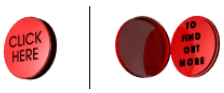


Figure 6: 2 frames of the dynamic button

Figure 7: Car on complex-dynamic background

The preferences for this part of the experiment were:

Preferred Blue	Preferred Purple
88%	12%

As can be seen from this subsection's experiment, if two objects are presented on the identical background and the subjects are given a task (in this case the question: "Which car do you prefer?") their visual behaviour is consistent with a top-down process and what happens outside the zone of interest is not visually important.

4.2 Comparison of different backgrounds

Now that we clearly established that the blue car is preferred over the purple one we needed to devise an experiment in which we could manipulate the environment so that subjects would no longer prefer the blue car over the purple one. We did so by using different contexts for the two objects. The hypothesis was that if the less preferred car was put in a more complex environment then it is likely that, as some subjects might prefer more complex images, just because they are more complex, they would thus choose the less preferred car. We performed this experiment in two stages.
In the first stage the subjects were asked to answer the question:

1. Which image do you prefer?

In the second stage the subjects were asked:

2. Which car do you prefer?

A different set of subjects was used for the two stages of the experiment. We thus wished to investigate whether, by introducing both bottom-up and top-down visual processes in images we could significantly influence subjects' preferences of objects. We used two pairs of images for this experiment. The first pair consisted of the blue car on neutral background and the purple car on the colourful background as in Figure 8. The second pair consisted again of the blue car on the neutral background and the purple car on colourful-dynamic background as shown in Figure 9. The hypothesis we are considering is: Even when subjects are given a task within an image (such as in question 2), if the context is sufficiently salient, the subject's preference could be significantly influenced.

Figure 8: Car on complex background

Figure 9: Car on complex-dynamic background

4.3 Question: Which image do you prefer?

These are the results of 16(8+8) subjects' preferences when presented a pair of images in figure 8.

 ISBN 978-80-86943-98-5

1. Comparison of static and complex backgrounds

Preferred Blue	Preferred Purple
37%	63%

The following are the results of subjects' preferences when presented a pair of images in figure 9.

2. Comparison of static and complex-dynamic backgrounds

Preferred Blue	Preferred Purple
25%	75%

As expected, when there was no clear task given within the image, the subjects preferred the more complex image dispate the fact the car that appears on this image is the car less preferred. During this question, subjects' perception worked in bottom-up fashion and most of the subjects were attracted to the more complex image.

4.4 Question: Which car do you prefer?

The following are the results of 18(9+9)subjects' preferences when presented a pair of images in figure 8.

1. Comparison of static and complex backgrounds

Preferred Blue	Preferred Purple
78%	22%

The following are the results of subjects' preferences when presented a pair of images in figure 9.

2. Comparison of static and complex-dynamic backgrounds

Preferred Blue	Preferred Purple
44%	56%

As we can see from the results above, the subjects responded in the top-down fashion with the first pair (figure 8) of images (their responses were task driven). Even though we introduced salient objects (the pictures) in the background, they were not salient enough to draw attention away from the task object (car). However, the experiment with the second pair images (figure 9) shows different results. With even more saliency added to the background (an opening window with a different text when the window is opened and closed) we were able to influence the subjects' preference of the task object itself (car). This suggests that we have successfully drawn attention away from the task region (car). Our test group in this case slightly preferred the purple car that before, in all being equal circumstances, was far less preferred than the blue car.

5 Conclusions and Future Work

In this paper we wished to explore how a viewer's preference of objects such as cars can be influenced by the overall appearance of realistic virtual environment. In particular we wanted to investigate whether the choice of the complexity of the background was sufficient to significantly influence the perception of the object of interest in the image.

Our results have shown that by only changing the background of objects to a complex and colourful one, we cannot significantly influence the perceptual preference of subjects. However, what we did discover is that other parameters of the image, in particular the presence of a highly salient animation in the background can significantly influence subjects' perception of the virtual environment.

So, it is possible to create an environment where, even though the subjects are performing a task in a specific region of the image, their overall perception of a targeted object can still influenced by what is happening outside of that specific region.

This is suggesting that the top-down process can be influenced in a bottom-up fashion.

This result has important implications for the design of virtual and augmented reality environments.

In future work, we will consider the effect that even more complex virtual environments may have on the perception of e-commerce objects and investigate whether we can use such salient cues to attract a viewer to consider "purchasing" additional objects in the virtual environment.

References

ARIS. 2003. Augmented reality image synthesis through illumination reconstruction and its integration in interactive and shared mobile ar-systems for e-(motion)-commerce applications proposal number: Ist-2000-28707. In *http://aris-ist.intranet.gr/*.

BOETTCHAR, L. C. 2005. No. one car color in america. In *http://channels.netscape.com/autos/package.jsp?name=fte/carcolors/carcolors*.

BOLIN, M., AND MEYER, G. 1998. A perceptually based adaptive sampling algorithm. In *SIGGRAPH 98 Conference Proceedings*, 299–310.

CATER, K., CHALMERS, A., AND WARD, G. 2003. Detail to attention: Exploiting visual tasks for selective rendering. In *Proceedings of the Eurographics Symposium on Rendering*, 270–280.

DALY, S. 1993. The visible differences predictor: An algorithm for the assessment of image fidelity. In *Digital Images and Human Vision*, MIT Press, Cambridge, Massachusetts, 179–206.

FERWERDA, J. A., PATTANAIK, S. N., SHIRLEY, P., AND GREENBERG, D. P. 1996. A model of visual adaptation for realistic image synthesis. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, 249–258.

HABER, J., MYSZKOWSKI, K., YAMAUCHI, H., AND SEIDEL, H.-P. 2001. Perceptually guided corrective splatting. In *Computer Graphics Forum*, vol. 20, 142–152.

HASIC, J., AND CHALMERS, A. 2006. Visual attention for influencing the perception of virtual environment. In *SCCG2006 Proceedings (a short paper)*.

 ISBN 978-80-86943-98-5

ITTI, L., AND KOCH, C. 2000. A saliency-based search mechanism for overt and covert shifts of visual attention. In *Vision Research*, vol. 40, 1489–1506.

JAMES, W. 1890. A saliency-based search mechanism for overt and covert shifts of visual attention. In *Principles of Psychology, New York: Holt*.

LUEBKE, D., AND HALLEN, B. 2001. Perceptually driven simplification for interactive rendering. In *Proceeding of 13h Eurographics Workshop on Rendering*, 221–223.

MACK, A., AND ROCK, I. 1998. Inattentional blindness. In *Proceedings of Symposium on Interactive 3D Graphics, Massachusetts Institute of Technology Press*.

MCNAMARA, A., CHALMERS, A. G., TROSCIANKO, T., AND GILCHRIST, I. 2000. Comparing real and synthetic scenes using human judgements of lightness. In *Proceeding of 12th Eurographics Workshop on Rendering*.

MYSZKOWSKI, K., TAWARA, T., AKAMINE, H., AND SEIDEL, H. P. 2001. Perception-guided global illumination solution for animation rendering. In *Proceedings of SIGGRAPH 2001, ACM Press/ACM SIGGRAPH, New York, E. Fiume Ed.*, 221–230.

MYSZKOWSKI, K. 1998. The visible difference predictor: Application to global illuminarion problems. In *Proceeding of Eurographics Workshop on Rendering, G. Drettakis and N. Max Eds.*, 223–236.

O'SULLIVAN, C., HOWLETT, S., MORVAN, Y., MCDONNELL, R., AND O'CONNOR, K. 2004. Perceptually adaptive graphics. In *STAR, Eurographics*, 141–164.

PATTANAIK, S. N., FERWERDA, J. A., FAIRCHILD, M. D., AND GREENBERG, D. P. 1998. A multiscale model of adaptation and spatial vision for realistic image display. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, 287–298.

RAMASUBRAMANIAN, M., PATTANAIK, S. N., AND GREENBERG, D. P. 1999. A perceptually based physical error metric for realistic image synthesis. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, 73–82.

REDDY, M. 1997. *Perceptually Modulated Level of Detail for Virtual Environments*. Ph.D. Thesis (CST- 134-97), University of Edinburgh.

SUNDSTEDT, V., CHALMERS, A., CATER, K., AND DEBATTISTA, K. 2004. Top-down visual attention for efficient rendering of task related scenes. In *VMV 2004 - Vision, Modelling and Visualization*.

WATSON, B., FRIEDMAN, A., AND MCGAFFEY, A. 2001. Measuring and predicting visual fidelity. In *Proceedings of SIGGRAPH 2001*, 213–220.

YARBUS, A. 1967. Eye movements during perception of complex objects. In *Eye Movements and Vision*, 171–196.

YEE, H., PATTANAIK, S., AND GREENBERG, D. 2001. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. In *ACM Transactions on Computer Graphics*, vol. 20, 39–65.

 ISBN 978-80-86943-98-5

Fast Colorization Using Edge and Gradient Constrains

Yao Li
Digital Media Lab,
Shanghai Jiao Tong university
No.800 Dongchuan Road
200240, Shanghai, China
yaolily77@sjtu.edu.cn

Ma Lizhuang
Digital Media Lab,
Shanghai Jiao Tong university
No.800 Dongchuan Road
200240, Shanghai, China
ma-lz@cs.sjtu.edu.cn

Wu Di
Electrical Engineering Department
Shanghai Jiao Tong university
No.800 Dongchuan Road
200240, Shanghai, China
guanliao@sjtu.edu.cn

ABSTRACT

This paper proposes a fast but effective user-guided colorization algorithm. The main difficulty with colorization is its intensive computational cost. And the color sometimes diffuses from one region to others. We make full use of the information given by the gray-scale images, including the edge, gradient and gradient direction, to propagate color over regions from the user's scribbles. We introduce a novel local distance definition which reduces the color confusion between two regions obviously. Two-Dimensional Programming is used to get the minimum distance from the scribbles to every pixel, and every pixel is blended by the chrominance with top three minimum distances. This speeds up the colorization process. Our algorithm can also be extended to recolorization and movie colorization. Experimental results show that our algorithm outperforms many state-of-the-art algorithms from small amount of user scribbles. Besides, the implementation require less than 1 second on the image with 320*240 pixels.

Keywords

Colorization, recolorization, interactive

1. INTRODUCTION

Color can be added to gray-scale images in order to increase the visual appeal of images such as old black and white photos, classic movies or scientific illustrations. In addition, the information content of some scientific images can be perceptually enhanced with color by exploiting variations in chromaticity as well as luminance.

Colorization is a term introduced by Wilson Markle in 1970 to describe the computer assisted process he invented for adding color to black and white movies[1]. The term is generically used now to describe the process of adding color to monochrome still images and movies. Colorization is the problem of assigning three-dimensional pixel values to an image which varies along only one dimension. Since different colors may have the same luminance value but vary in hue or saturation, the problem of colorizing gray-scale images has no inherently "correct" solution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

The main difficulty with colorization is its intensive computational cost. The user cannot see the effect immediately and colorize the images interactively. Levin[2] propose a new user-guided colorization technique based on the premise that nearby pixels in space that have similar gray levels should also have similar colors. However, because no boundary or region information is considered in their colorization process, unseemly color sometimes diffuses from one region to others. The processing time of colorizing an image with 320*240 pixels costs ten or more seconds.

In this paper, we propose a user-guided colorization method inspired by Levin's. This paper makes several specific technical contributions. First, we introduce a novel local distance definition, which take the edge, gradient and gradient direction into accounts. With the edge and gradient constrains, the color confusion between two regions reduce obviously. Next, we use Two-Dimensional Programming to get the minimum distance from the scribble to every pixel. This make the process time

only one tenth of Levin's. We also extend our algorithm to movies.

2. RELATED WORK

Colorization has been extensively studied in the movie industry since 1970's. Various analogue techniques have been used to accomplish this challenging task[3]. In this section we focus on digital colorization only.

Semi-automatic Colorization

Welsh[4] exploited local luminance distribution as textural information and transfer the color between the similar texture regions in the source color image and the target gray-scale image. The idea is to transfer color from neighborhoods in the reference image that match the luminance in the target data. There is then an underlying assumption that different colored regions give rise to distinct luminance, and their approach works properly only when this is not violated, otherwise requiring significant user intervention. This approach is a special case of the more general image analogies framework[5]. The results reported by the authors are quite impressive, although the technique intrinsically depends on the user to find proper reference data.

Irony[6] presented a method to colorize grayscale images by transferring color from a segmented example image. Instead of relying on a series of independent pixel-level decisions, they developed a new strategy that attempts to account for the higher-level context of each pixel. Target pixels are classified using LDA and image space voting with DCT coefficients as feature vectors. The colorized results exhibit nice spatial consistency.

Sykora[7] introduced a colorization framework for old black-and-white cartoon video. The dynamic part of the scene is represented by a set of outlined homogeneous regions which superimpose the static background. They combined unsupervised image segmentation, background reconstruction, and structural prediction to reduce manual intervention.

User-guided Colorization

Levin[2] proposed a simple yet effective user-guided colorization method. In this method the user is required to scribble the desired colors in the interiors of the various regions. These constraints are formulated as a least-squares optimization problem that automatically propagates the scribbled colors to produce a completely colorized image. The basic premise is that neighboring pixels having similar intensities in the monochrome data should have similar colors in the chroma channels .Other algorithms based on color scribbles have subsequently been proposed [8][9]. These approaches have produced some impressive colorizations from a small amount of user input.

Manga[10] Colorization proposed a novel colorization technique that propagates color over regions exhibiting pattern-continuity as well as intensity continuity. It propagates colors over pattern-continuous regions containing hand drawn hatching and printed screening patterns. The user is free from manually segmenting the patterned regions. Both pattern-continuous and intensity-continuous regions can be segmented under the same mathematical framework.

3. ALGORITHM

Our colorization method is working in YUV color space(other color spaces could be used as well). The problem of colorization is to estimate the U, V component value of the input gray-scale image from the only known intensity value Y. That is an ill-posed problem in which two and more solutions exist, since different color may have the same intensity value.

Our algorithm is based on three assumptions:

- The colors have high correlativity between pixels in one local region and low correlativity in different regions;
- Pixels having similar intensities have similar colors;
- The color values in the input image change smoothly.

We start with the description of the proposed algorithm for still images. Let $Y(x,y):\Omega \rightarrow R^+$ be the given monochromatic image defined on the region. The given monochromatic image becomes the luminance Y. The goal is to compute $U(x,y):\Omega \rightarrow R^+$ and $V(x,y):\Omega \rightarrow R^+$. The colors are given in a region Ω_c, and $|\Omega| << |\Omega_c|$. This information is provided by the user via color strokes in editing type of applications, or automatically obtained for compression (selected compressed regions) or wireless (non lost and transmitted blocks) applications. The goal is from the knowledge of Y in Ω and U, V in Ω_c to get the color information (U,V) into the rest of Ω, Ω-Ω_c.

$P_{s,t}$ represent a path connecting s and t, where $s \in \Omega_c$ and $t \in \Omega$-Ω_c. Our objective is to find a path which have the minimum link cost from each scribble to t. We define the geodesic distance between s and t by:

$$dist(s,t) = \min_{P_{s,t}} \sum J(p,q) ,$$

$p,q \in P_{s,t}$, $J(p,q)$ is the local link cost between two neighbor point.

 ISBN 978-80-86943-98-5

Every scribble gives a uniform color c. To every point t, we define the geodesic distance $d_c(t)$ as the minimum distance $d(s,t)$ from t to s of the same chrominance c:

$$dist_c(t) = \min_{\forall s \in \Omega_c} d(s,t)$$

Local cost

Figure 1 shows the result of Levin[2]. At first glance, the result is exciting. But there is color confusion near the edge between two images even after careful color choosing and scribbling(figure 1.c). If we change the color of the scribbles, we can see that there are obvious color confusion near the area between two regions(figure 1.d).

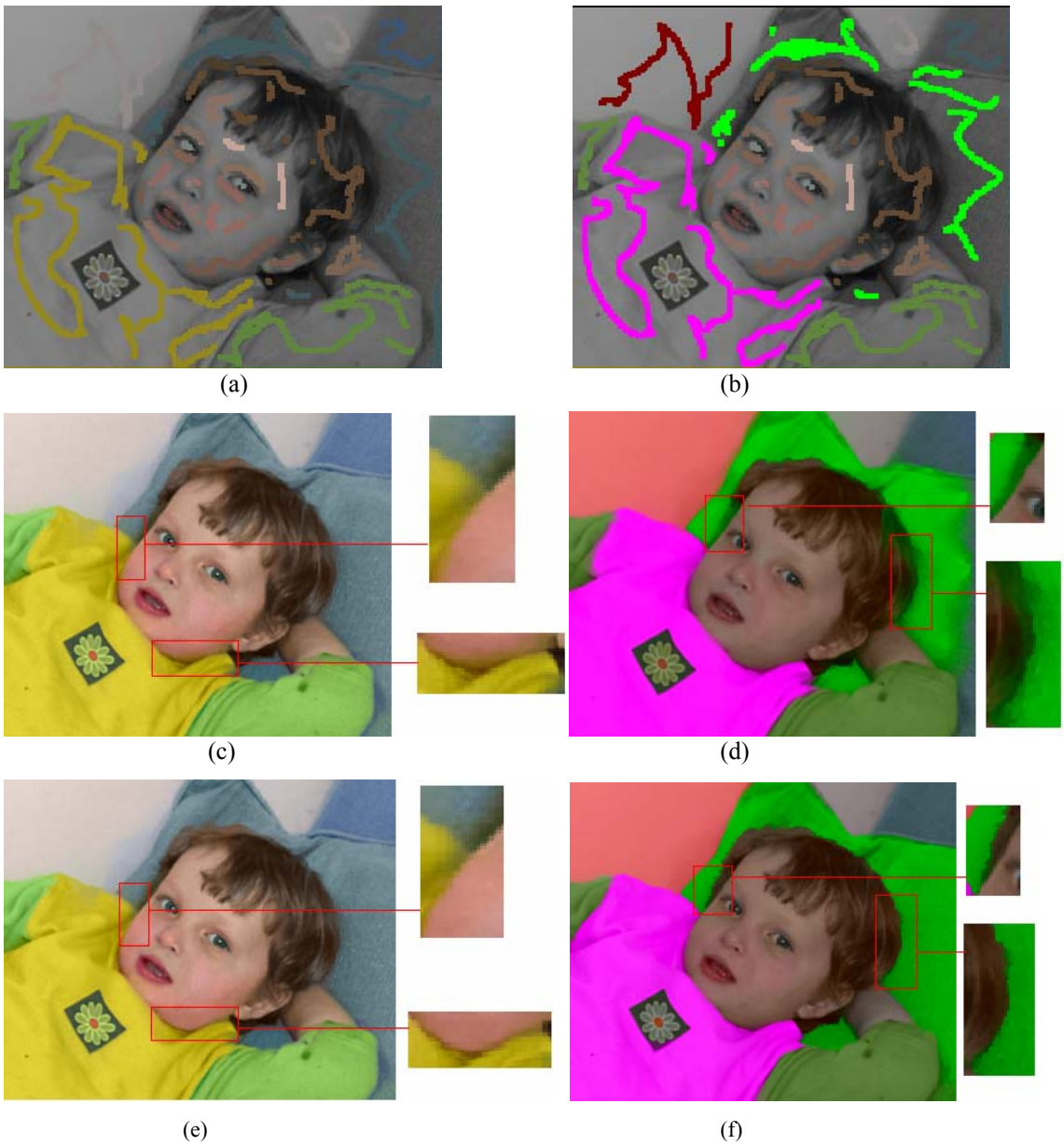


Figure 1. Comparison of Levin's results[2] and ours. (a) The gray-scale image with color scribbles from [2], (b)The gray-scale image with color scribbles whose positions are same as(a), but color changes; c) and (d) are the results using Levin's algorithm; e)and f) are the results of our algorithm.

The edges, the gradient and its direction give important cue of the color, so we compute the local cost of two neighbor point as a weighted sum of three parts:

$$J(p,q) = w_E J_E(p,q) + w_G J_G(q) + w_D J_D(p,q),$$

where J_E is the edge cost, J_G is the gradient cost and J_D is the gradient direction cost.

 ISBN 978-80-86943-98-5

We use Canny edge detector to extract the boundary of the input gray image I and restore it into the boundary image I_E, then

$$J_E(p,q)=\begin{cases}1; & if\ \ p\in I_E\ \ or\ \ q\in I_E\\ 0; & otherwise\end{cases}$$

Then, we calculate every pixel's gradient magnitude:

$$G=\sqrt{I_x^{\ 2}+I_y^{\ 2}}\ ,$$

where I_x and I_y represent the partials of an image I in x and y respectively.

To keep the resulting maximum gradient at unity, we define:

$$J_G=\frac{G}{\max(G)}$$

Then $J_G(q)$ is scaled by 1 if q is a diagonal neighbor to p and by $\sqrt{2}$ if q is a horizontal or vertical neighbors.

Next, we define the gradient direction cost. For every pixel, the cost is computed in 8 directions respectively. The gradient in each direction is defined as:

$$J_i(p,q)=\begin{cases}\dfrac{q_{i+1}-q_{i-1}}{2}, i \text{ is the diagonal direction}\\ \dfrac{q_{i+1}+q_{i+2}-q_{i-1}-q_{i-2}}{4}, \text{ otherwise}\end{cases},$$

where $q\in N(p)$, q_{i+j} means the neighbor in the j clock wise direction p, q_{i+j} means the neighbor in the j counter clockwise direction to p, $i\in(0,8)$, $j\in(-2,2)$.

Color blending

We compute the U, V component of every point $t\in\Omega-\Omega_c$ by blending the different chrominance in Ω_c:

$$C(t)=\frac{\sum_{\forall c\in\Omega_c} W(d_c(t))c}{\sum_{\forall c\in\Omega_c} W(d_c(t))}, \qquad (1)$$

where $W(.)$ is a blending weight function. Here we use $w(r)=e^{-r^2/2}$.

Obviously, the chrominance with lower intrinsic distance effect the point's color more. We found that we can get the satisfactory results using three closest chrominance to blend the color. So we only use the chrominance with the three closest distances to blend the point. This can fast the blending speed.

Two-Dimensional Programming

The pixels with link cost construct a graph. Dynamic programming can be formulated as a directed graph search for an optimal path. We utilizes an optimal graph search similar to that present by Dijkstra[11] and Nilsson[12].

Firstly, we construct a heap to store the nodes sorted by link cost. The user's scribbled colors are the seeds and we insert them into the heap. Then an expending process deals with all their neighbors and inserts them to the heap until the end point is reached.

The *2-D* dynamic programming (DP) graph search algorithm is follows:

Input :

I //Grayscale image

s //scribbles drawn by user

Output:

I' //colored image

Data Structures:

M; // a map structure which store the chrominance reached this point and the correspondence distance.

blend;//the color blend result of a point

P(p);// a structure store a point p's M and blend

Link; // a path whose nodes pointer the point/pixel

I_E; //The edge image;

Algorithm:

Using Canny edge detector to get edge image I_E;

For $t\in\Omega$

New *P(t)*;//Create an empty *M* for every point;

End for

New *Link_pool*;

For $p\in\Omega_c$

P(p).M.push(*c*,0); //set the point in the scribbles to be the chrominance of the scribbles; and the distance to be *0*;

Link.push(*P*(*p*));

End for

While *Link* is not empty

For all *P*, Get the pair (c_p,$dist_p$)with smallest distance

Pop *P* from *Link*;

For all $q\in N(p)$

Get *P(q)*;

Calculate dist*(p,q)*;

$dist_q = dist_p + dist(p,q)$;

P(q).M.push(pair(c_p,c_q));

If(*P(q).M.pair_no*>=3

P(q).M.sort;

P(q).M.pop_back;//only remain the three smallest distance

End if

Link.push*(P(q))*;

 ISBN 978-80-86943-98-5

End for

For $t \in \Omega$

Caculate $P(t)$.blend equation(1)

End for

Video

Levin[2] uses optical flow tracking to propagate colors across time. We avoid this explicit computation. We opt for a simpler formulation. Following the color constancy constraint often assumed in optical flow, and if the gradient fields and motion vectors of all the movie channels are the same, then of course we can consider:

$$\frac{\partial Y}{\partial t} = \frac{\partial U}{\partial t} = \frac{\partial V}{\partial t},$$

where t is the time coordinate in the movie.

Recolorization

Recolorization is to replace the colors of an image by new colors. The original color in the image give us more cue than the gray images. We assume that the color changes indicate the new color changes. That's to say, in the recolorization process, the original chrominance give more information than intensity. So we can replace the Y by U or V in equation (1).

4. RESULTS AND ANALYZE

Our blending algorithm's time and space complexity is $O(|\Omega|\cdot|c|)$, where c is the chrominance of Ω_c. Because $|c|$ is small and we only use the three chrominance with lowest distance to blend. So the time and space complexity are reduced to $O(|\Omega|)$.

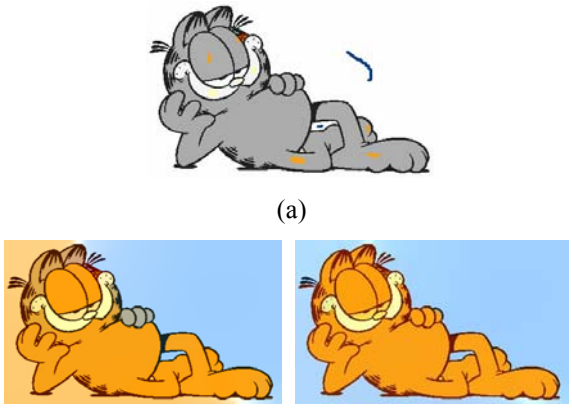
The colorization system is built on a Pentium IV 1.9Ghz CPU and 512M RAM. Most images can be colorized reasonably well in under a second. We get the scribbled pictured from Levin's website and using our algorithm on them. As figure 2 shows, we get almost the same satisfied results as Levin's. Levin takes more than ten seconds, while our algorithm only takes less than one second (Table 1). Figure 1.c and d is our results. From the insets, we can see there are less confusion around the face and hair. We give more results on nature scenes in figure 5.

Because we take edges into accounts, our proposed algorithm needs smaller scribbles than [2], especially for the images with strong edges, such as cartoons. In figure 3, our algorithm colorizes the gray-scale Garfield good. Figure 4 shows more results on natural scene. Our algorithm also gives good results. Garfield's right and left area are colorized correctly because of their connection.

Figure 2 The first column is the gray images marked with color scribbles from [2]. The second column is our colorization results.

	Pixels	Time in [2] (second)	Ours (second)
Cats	319*267	15.209	0.712
Girl	318*238	10.128	0.421
Building	399*299	15.267	0.917

Table 1 The comparison of the processing time between ours and Levin's under the same platform.



(a)

(b) (c)

Figure 3 Colorization of Garfield. (a) The gray-scale image with color scribbles, (b) The colorization using algorithm in[2] (c) Our colorization result.

 ISBN 978-80-86943-98-5

Figure 5 shows how our algorithm can be applied to recolorization. There is no need to mask the object as [2]. What the user need is to scribble the color on the areas where he want change the color. Our algorithm can recolor the object according to their original color change with the edge and gradient constrains.

Figure 6 shows 8 frames from the movie "Ice Age 2". We change their color to gray to test our algorithm. We only scribble color on one frame. The remaining frames can be colorized around 2 seconds. And the effect is good.

Figure 4: More still examples. Right column: the input images with scribbled colors. Left column: colored images.

5. SUMMARY

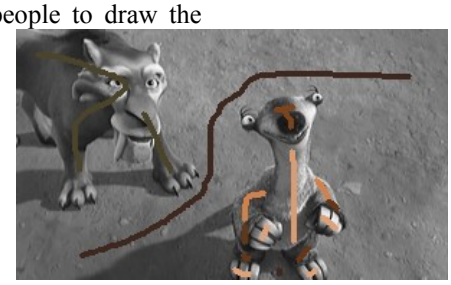
A simple, effective and fast colorization framework was introduced in this paper. While keeping the quality at least as good as state-of-art algorithm, our propose method colorize images within a second. Besides the improvement on speed, our algorithm reduces the color confusion near the boundary by bringing in edge and gradient cost. Our algorithm also can be applied on movie efficiently.

There are many important topics to explore in the future work. First, it is hard to people to draw the scribble in a right region to make the algorithm more efficient. So we need to find a way to understand what kind of information is needed in the chroma channels for error controlled colorization. This will be helpful in editing images, directing the user to the crucial regions to provide the strokes. Second, we can use the colorization onto compression. Only a few color samples can colorize the entire gray-scale image. This can be explored to a new color image compress method. Third, this colorization algorithm can be used on wireless image transmission. When some data lost in transformation process, using the relation between the chrominance and intensity, the lost data can be repaired.

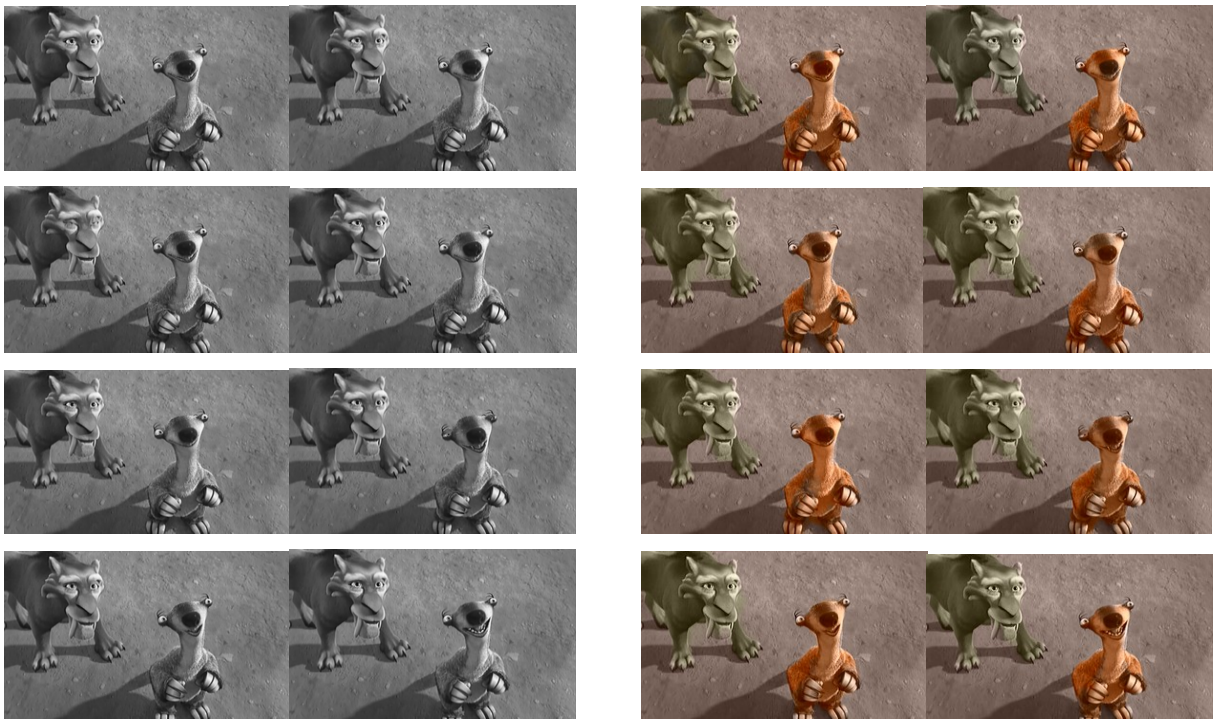
Figure 5 Recolorization results. The first row is the original images, the second row is the images with color scribbles, and the last row is the result.

6. ACKNOWLEDGMENTS

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2006CB303105 and the National Natural Science Foundation of China under Grant Nos. 60373070.



(a)



(b) (c)

Figure 6 Colorizing the movie by our colorization algorithm. (a) The first frame with the color scribbles. (b) The 8 gray-scale frames. (c) Colorization results.

7. REFERENCES

[1] G. Burns, Colorization. Museum of Broadcast Communication:Encyclopedia of Television. Http://www.museum.tv/archives/etv/index.html

[2] Levin A., Lischinski D., and Weiss Y. 2004. Colorization using optimization. ACM Transactions on Graphics, 3, 689–694.

[3] Markle. W. 1984. The development and application of colorization. SMPTE Journal, 632–635.

[4] Welsh T., Ashikhmin M., and Mueller K. 2002. Transferring color to grayscale images. ACM Transaction on Graphics 21,3(July),277-280

[5] Hwerzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H.: Image analogies. In Proc. SIGGRAPH, 2001 (Aug. 2001), Computer Graphics Proceedings, Annual Conference Series, pp. 327–340.

[6] Irony, R., Cohen-or, D., and Lischinski, D. 2005. Colorization by example. In Proc. of Eurographics Symposium on Rendering 2005, 201–210.

[7] Sykora, D., Burianek, J., and Jirizara. 2004. Unsupervised colorization of black-and-white cartoons. In Proc. of Third International Symposium on Non Photorealistic Animation and Rendering, 121–128.

[8] Sapiro G.: Inpainting the colors. IMA Preprint Series, Institute for Mathematics and its Applications University of Minnesota, 1979 (2004).

[9] Yatziv L., Sapiro G.: Fast image and video colorization using chrominance blending. IMA Preprint Series, Institute for Mathematics and its Applications University of Minnesota, 2010 (2004).

[10] Yingge Qu, Tien-Tsin Wong, Phen-Ann Heng, Manga Colorization, In Proc. SIGGRAPH 2006, Computer Graphics Proceedings, Annual Conference Series pp: 1214 - 1220.

[11] E.W.Dijkstra,"A note on Two problems in Connexion with Graphs" numerische Mathematik, vol1.1, pp.269-270, 1959

[12] N.J.Nilsson, Principles of Artificial Intelligence. Palo Alto, CA: Tioga, 1980.

 ISBN 978-80-86943-98-5

SIFT Implementation and Optimization for General-Purpose GPU

S. Heymann
Fraunhofer HHI
Einsteinufer 37
Germany 10587, Berlin
heimie@selective.de

K. Müller
Fraunhofer HHI
Einsteinufer 37
Germany 10587, Berlin
kmueller@hhi.de

A.Smolic
Fraunhofer HHI
Einsteinufer 37
Germany 10587, Berlin
smolic@hhi.de

B. Fröhlich
Bauhaus University Weimar
Fakultät Medien
Germany 99423, Weimar
bernd.froehlich@medien.uni-weimar.de

T. Wiegand
Fraunhofer HHI
Einsteinufer 37
Germany 10587, Berlin
wiegand@hhi.de

ABSTRACT

With the addition of free programmable components to modern graphics hardware, graphics processing units (GPUs) become increasingly interesting for general purpose computations, especially due to utilizing parallel buffer processing. In this paper we present methods and techniques that take advantage of modern graphics hardware for real-time tracking and recognition of feature-points. The focus lies on the generation of feature vectors from input images in the various stages. For the generation of feature-vectors the Scale Invariant Feature Transform (SIFT) method [Low04a] is used due to its high stability against rotation, scale and lighting condition changes of the processed images. We present results of the various stages for feature vector generation of our GPU implementation and compare it to the CPU version of the SIFT algorithm. The approach works well on Geforce6 series graphics board and above and takes advantage of new hardware features, e.g. dynamic branching and multiple render targets (MRT) in the fragment processor [KF05]. With the presented methods feature-tracking with real time frame rates can be achieved on the GPU and meanwhile the CPU can be used for other tasks.

Keywords

GPU, SIFT, feature extraction, tracking.

1. INTRODUCTION

With the inclusion of programmable parts in modern graphics hardware, such as vertex and fragment processors, developers started using the power of graphics processing units (GPUs) for general purpose computations beyond creating beautiful pictures and creating high-end game engines. The research field that arose from those efforts is known as GPGPU (general purpose computations on GPUs). To make use of the GPU for more general computations it is necessary to transform the algorithms under investigation such that it optimally utilizes the parallel processing model used on modern graphics hardware. A number of algorithms and applications have been implemented onto GPGPUs, e.g. kd-tree search [FS05], sorting algorithm acceleration [GHLM05] and database search [GLW*04]. An overview of concepts for algorithm transformation to GPU architecture is given in [Har05] and [OLG*05].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright UNION Agency – Science Press, Plzen, Czech Republic.

In this paper we show, how a feature extraction algorithm can be adapted to make use of modern graphics hardware and which processing acceleration can be obtained by optimizing all stages of the algorithm. Specifically, the SIFT Algorithm [Low04a] was implemented on a NVIDIA QuadroFX 3400 GPU with 256MB video RAM, taking advantage of new functionalities, e.g. dynamic branching and multiple render targets (MRTs) of the

 ISBN 978-80-86943-98-5

fragment and vertex processor. After reviewing the related work, we give a short overview of the algorithm, followed by the stage-by-stage GPU implementation and the obtained overall acceleration of the algorithm in comparison to standard CPU implementation.

2. RELATED WORK

One of the main problems of computer vision is the generation of stable feature-points from natural images. These feature-points are used for correspondence matching to find known objects and gain information about their presence, position, size or rotation in other images [BL02]. One approach for feature tracking is given in [Fun05], but the extracted feature points were not invariant to scale and the application was focused on the usage of multiple parallel graphics cards. One method to create highly stable feature-vectors from images is the Scale Invariant Feature Transform (SIFT) introduced by David Lowe [Low04a]. SIFT features are invariant against rotation, changes in scale and lighting/contrast and can therefore be well applied to scene modeling, recognition and tracking [GL04] and panorama creation.

3. SIFT OVERVIEW

The SIFT method consists of different stages to obtain relevant feature points. These stages where analyzed and individually adapted to maximize GPU parallel processing using only few CPU accesses. The single SIFT stages are:

1. Search for potential points of interest by creation of a Difference of Gaussian (DoG) scale-space pyramid as image representation and filtering for extreme values
2. Further filtering and reduction of the obtained points from 1. to select stable points with high contrast. To each remaining point, its position and size are assigned.
3. Orientation assignment to each point by finding a characteristic direction.
4. Feature vector calculation based on the characteristic direction from 3. to provide rotation invariance.
5. The whole process is stacked in a way that only a subset of elements from the beginning of a stage is passed onto the next stage.

To achieve scale invariance it is necessary to create a representation of the image frequencies. This is realized using a scale space pyramid as introduced by Witkin [Wit83]. Each image within the pyramid refers to different image frequencies. By searching in all images of the scale space pyramid, the obtained feature point candidates become scale invariant. The scale-space pyramid is constructed by taking a gray-scaled version of the original image and convolving it repeatedly with Gaussian convolution kernels of increasing size. Thus a number of images with increasing blurriness is constructed as shown in the stack of four 640x480 images in Figure 1 top-left. In the next stage, the most blurred image from this stack is downscaled by a factor of two and afterwards convolved with the same set of Gaussian kernels as before to create the next stack of four 320x240 images. The whole process is repeated until a specified size has been reached, which is 80x60 in the example in Figure 1 top-right.

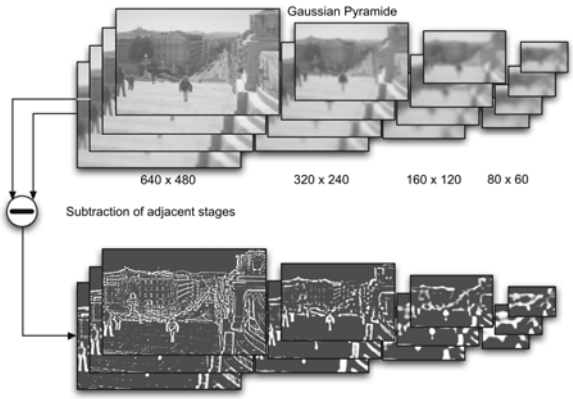


Figure 1. Gaussian pyramid (top) and difference of Gaussian (bottom).

Now the pyramid consists of continuously convolved versions of the original image with different sizes and blurriness. To calculate the single frequencies of the image, adjacent images or stages of the same size of the pyramid have to be subtracted to create the Difference of Gaussian (DoG) representation, as shown in Figure 1 bottom. Finally, the obtained DoG pyramid is filtered to find the global extreme values. The filtering is applied pixel wise by comparing the luminance of the current pixel to its 8 neighbors within the same image, as well as to its 9 non-shifted neighbors of both adjacent layers of the same size. If the luminance valueof the pixel under investigation is a minimum or maximum among all these neighboring luminance values, the pixel is considered as feature point candidate.

In the next step, the obtained candidates are further filtered to eliminate feature points that are unsuitable for correspondence detection. Here, mainly two types of unsuitable points are considered. First, points erroneously found due to noise in the input image and second points that lay on edges. The first type is eliminated by introducing a threshold for luminance differences between a possible feature point and its neighbors. Only, if the threshold is exceeded, the

 ISBN 978-80-86943-98-5

point is further processed and considered as candidates. Edge points need to be excluded, since they are unsuitable for tracking and correspondence matching. For edge point detection, the surface curvature around the surface D(x,y) of a candidate point at position (x,y) can be analyzed using the Hessian Matrix $\boldsymbol{H}$ of second order local derivatives:

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 D(x,y)}{\partial x^2} & \frac{\partial^2 D(x,y)}{\partial x \partial y} \\ \frac{\partial^2 D(x,y)}{\partial x \partial y} & \frac{\partial^2 D(x,y)}{\partial y^2} \end{bmatrix}. \quad (1)$$

As shown by Harris and Stephens [HS88], the curvature of D(x,y) is proportional to the Eigen vectors of $\boldsymbol{H}$. Since we are only interested in a criterion for edge or non-edge points, only the ratio between both Eigen values e1 and e2 with $e_1 \geq e_2$ is important. Let $r = e_1/e_2$ be this ratio, then:

$$\begin{aligned} \mathrm{trace}(\boldsymbol{H}) &= e_1 + e_2 = (r+1)e_2 \text{ and} \\ \det(\boldsymbol{H}) &= e_1 e_2 = r e_2^2. \end{aligned} \quad (2)$$

From this, the criterion for non-edge points is derived as:

$$\frac{\mathrm{trace}(\boldsymbol{H})^2}{\det(\boldsymbol{H})} < \frac{(r+1)^2}{r}. \quad (3)$$

In Lowe [Low04b], best results for excluding edge points have been reported for $r = 10$.

In the third step, the remaining feature points are assigned with their main orientation to achieve rotation invariance. Therefore, the gradients within a certain distance around each feature point are transformed into polar coordinates:

$$\begin{aligned} m(x,y) &= \sqrt{\left(\frac{\partial D(x,y)}{\partial x}\right)^2 + \left(\frac{\partial D(x,y)}{\partial y}\right)^2}, \\ \theta(x,y) &= \tan^{-1}\left(\frac{\frac{\partial D(x,y)}{\partial y}}{\frac{\partial D(x,y)}{\partial x}}\right). \end{aligned} \quad (4)$$

Then, a histogram is constructed from the phase values $\theta(x,y)$ and weighted with magnitude values $m(x,y)$, to obtain the main direction of the gradients around the feature points.

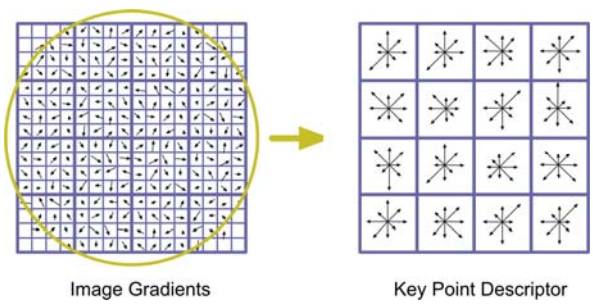


Figure 2. Image gradient transformation into key point descriptors.

Finally, feature vectors are created from the gradients. In Figure 2 left, the image gradients are shown. For each 4x4 region, the main orientation is used as new local coordinate system, meaning that also new texture coordinates need to be interpolated at intermediate positions. The 16 gradients within that particular region are than obtained within this new texture coordinate system. The created descriptor elements are a projection of the 16 gradients onto 8 directions, aligned in the local coordinate system, defined by the main orientation, as shown in Figure 2, right. This projection is carried out for all 4x4 regions, thus creating a feature vector of 128 entries. Finally, the feature vector is normalized to achieve invariance to contrast changes.

4. GPU-IMPLEMENTATION

For the implementation of the SIFT algorithm on the GPU, adaptation of the initial CPU algorithms were required to fit the algorithm well into the graphics pipeline [Zel05] and take full advantage of the GPUs parallel processing abilities. Therefore, the main focus was to restructure the different SIFT stages to fit the GPU texture format. The experiments where carried out on a system, using an Intel Xeon 3.2GHz CPU, 2GB of RAM and a NVIDIA QuadroFX 3400 GPU with 256MB video RAM and PCI Express x16 graphics bus.

4.1. DoG Pyramid Creation

SIFT features are generated from gray level images, whereas GPU texture buffers are designed for three color + one alpha channels. GPUs do not only have a parallel processing ability on a per pixel basis parallelized by the number of fragment processors on the GPU, there also is a parallelism on the computational stages of the GPU calculating the four color values at once as a vector. Having only gray images the computations done to convolve an image would waste 75% of the processing power. To make full usage of the vector abilities of GPUs the gray-level input image is modified. Here, we rearranged the gray image data into a four channel RGBA image, as shown in Figure 3.

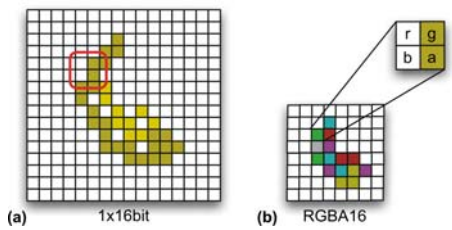


Figure 3. Texture packing to RGBA16 GPU format.

One color value in the RGBA image represents 2x2 pixel of the gray-level image, thus reducing the image area by 4. With the RGBA image, the

 ISBN 978-80-86943-98-5

convolution can be processed without wasting computational power on the GPU. In the case of a convolution, the processing on the packed data is straightforward, since here mostly linear operations, such as pixel- wise additions or multiplications are applied. In cases of operations, where pixel processing also depends on neighboring pixels, the algorithm adaptation for packed data becomes complicated, since all neighboring references need to be redirected. Reorganizing the input image creates some computational overhead which is comparatively low since the data remains in this packed format for the whole process of scale-space and DoG pyramid creation. The packing is implemented using a simple fragment shader that takes a block of 2x2 adjacent pixels and arranges them into one RGBA pixel. The Gaussian convolution is directly applied onto the packed RGBA format, as shown in Figure 4 with a 9tap Gaussian kernel. Here, the Gaussian kernel is split into even and odd values to carry out two separate semi-convolutions, which are added afterwards for the final result. Each pixel of the Gaussian kernel is multiplied with all four color components in one calculation, thus only requiring one texture access. The calculations with even and odd Gaussian kernel pixels are implemented in the same fragment shader and therefore the same texture access can be used for both steps.

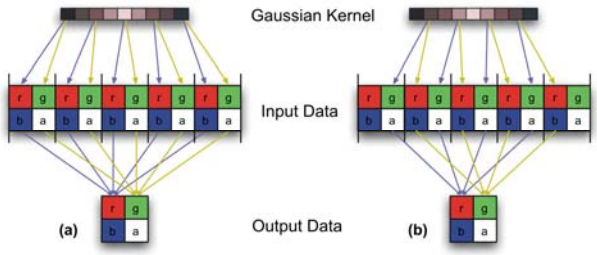


Figure 4. Gaussian convolution in RGBA16 GPU format, (a) odd and (b) even samples.

Horizontal and vertical filtering with Gaussian kernels of different sizes is applied successively and the differently blurred images are subtracted to create the DoG pyramid, described above.

Using this technique allows us to convert a color image into a gray level image and pack the pixels in the described way in one rendering pass.

4.2. Key Point Filtering and Orientation

For the detection of feature points, as described before, dynamic branching is used to keep the whole selection process in the GPU. Therefore, the criteria for possible feature points where rearranged starting with the luminance difference threshold, which excludes 50% of possible feature points. Then the search for global extreme values first compares a point with its 8 neighbors within the same buffer, leaving only 0.6% of possible points followed by comparison with the 9 pixel of the adjacent buffers within the DoG stack. Possible feature points are shown in Figure 5(b). Afterwards, the exclusion of noise and edge points is carried out, leaving stable feature points, as shown in Figure 5(c).

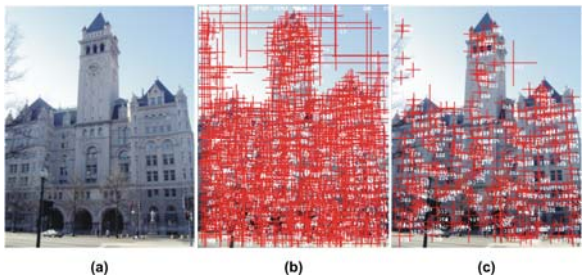


Figure 5. Extraction and filtering of features.

After filtering and localization of potential feature points, the corresponding feature vectors are calculated. To calculate the gradient direction and magnitude, MRT functionality is used. For both values, only the four direct neighboring pixels are required, which keeps the referencing for calculation relatively simple. The reqired pixel access and operations are shown in Figure 6.

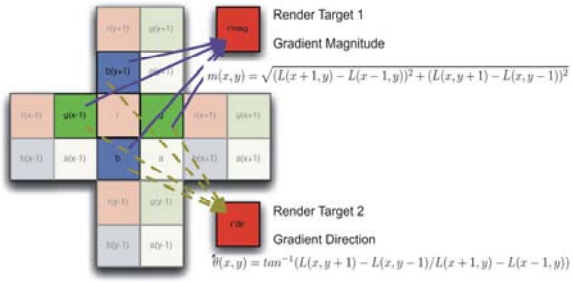


Figure 6. Gradient magnitude (*top*) and direction (*bottom*) calculation for red channel.

Here, the central texel “rgba” with its packed 4 original pixels as the four color components is processed. The magnitude and direction calculation for the central “r” component are shown in Figure 6, which require the four neighboring components that are highlighted as solid colors. Each central component requires two other components from the central texel and two from adjacent texels. The required operations for magnitude and direction calculation are also shown in Figure 6. Both calculations require the same input data. Since the input data is already packed, the use of two color components for magnitude and direction respectively is not possible. Instead, the use of MRTs greatly accelerates the processing, since both calculations can be carried out at once, writing the results into two separate rendering targets. Thus, time consuming *OpenGL* context switching is avoided and only one texel access for both operations is required, since

 ISBN 978-80-86943-98-5

magnitude and direction use the same intermediate calculation (i.e. horizontal and vertical subtraction).

4.3. Feature Descriptor Creation

Both render targets are now used to create the weighted histograms of the 4x4 regions around each feature point, as shown in the theoretical part in Figure 2. Each region is associated with 8 directions, adding up to a 128-element output vector. This operation differs from previously implemented operations, since for each single input element (or extreme point) 128 output elements are created. This operation can not be carried out at once, even with MRTs. Therefore, each region is processed in one fragment shader call, as there is no possibility to split the histogram calculation itself. For this calculation, it is useful to select a structure, where for each fragment different data can be accessed. A simple rectangular area, as used for the other calculations is not sufficient, since interpolation algorithms would interpolate the four corner attributes across the area, whereas here, each point requires independent attributes. A suitable representation for such independent attribute purposes is a vertex grid that can be created from geometry points via *glVertex2f()* or a line of multiple segments.

For easier processing, magnitude and direction values are rearranged from the two rendering targets into one texture to further process them with the precalculated texture directions, as shown in Figure 7.

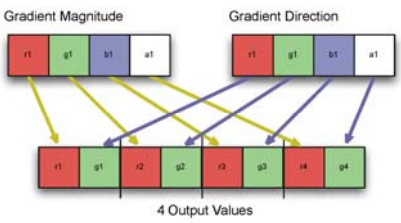


Figure 7. Gradient map unpacking into one texture.

Here, the packed values for magnitude and direction are unpacked and interleaved at the same time, such that one output value only contains one magnitude and one associated direction value. In this form, both values are contained in one texture that can be further processed without format change.

For the final feature creation, gradient histograms for the 4x4 areas of each extreme point are created. Each area has to be processed in one fragment shader call, since the histogram calculation itself cannot be split up without expensive calculations. To carry out the histogram calculation in one shader cycle, 8 output values have to be calculated simultaneously. This again can be achieved, using MRTs on advanced graphics cards. In Figure 8(a) the data structure for the feature generation is shown. As an example, a feature vector is shown in Figure 8(b), which consists of 16 vertices and is mapped into the two render targets.

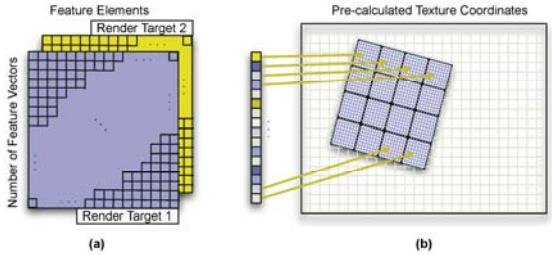


Figure 8. Render Targets for Feature Generation. (a) Frame buffers in both render targets and (b) feature point position in image and access on pre-calculated texture coordinates.

Each vertex of a feature vector is associated with appropriate attributes in the CPU. These attributes contain relative texture coordinates, magnitude and direction of gradient areas. The corresponding calculations can be carried out independently and all necessary parameters are coded in the feature vectors. As a result, each render target from Figure 8(a) contains a complete data set for half the feature vectors.

These SIFT feature vectors can now be used for correspondence matching between different images, e.g. for tracking in an image sequence. For this purpose, the Euclidean distance D between two feature vectors $V1$ and $V2$ with length N is calculated, as shown in (5).

$$D = \sqrt{\sum_{i=1}^{N} (V1_i - V2_i)^2} \ . \qquad (5)$$

In our SIFT implementation, each vector has $N = 128$ elements. The associated subtractions in (5) can be well calculated in parallel. In a test, a CPU and GPU implementation were compared in terms of processing time. For this test, a number of feature vectors K were taken, with K varying between 500 and 3000. Each feature vector was compared with each other, resulting in K^2 comparisons. While the processing time for 3000x3000 comparisons was 13sec using the CPU implementation, the GPU implementation only required 0.5sec.

4.4. Results

After optimizing all SIFT stages for efficient GPU processing, the entire algorithm was tested and compared against the original CPU implementation

 ISBN 978-80-86943-98-5

and a manually SSE (Streaming SIMD Extension) optimized version. The results are shown in Figure 9.

Here, the manually SSE optimized version requires 0.312sec compared to 0.406sec for the original CPU implementation. In comparison to that, the SIFT algorithm could be drastically accelerated by utilizing massive parallel GPU processing and thus achieving a processing time of only 0.058sec. Thus, SIFT feature extraction can be carried out in real time at approximately 20 frames/sec.

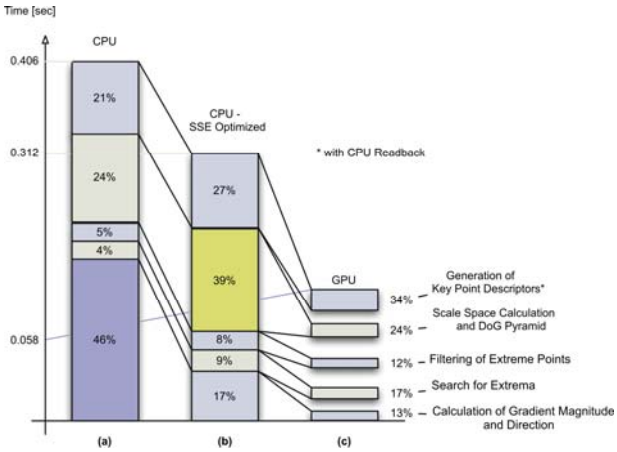


Figure 9. Results for all SIFT operator stages of the GPU implementation in comparison to standard CPU processing.

5. CONCLUSION

In this paper we have shown, how the SIFT algorithm can considerably be accelerated by utilizing GPU parallel processing. After arranging the luminance values of the input image into the GPU texture buffers RGBA format, all follow-up operations have also been adapted to this texture format and make use of new GPU technology, namely dynamic branching for the detection of relevant feature points and MRTs for parallel gradient direction and magnitude calculation. As a result, the SIFT algorithm can be applied to image sequences with 640x480 pixels at 20 frames/sec. Future work will mainly focus on real-time applications using SIFT features, e.g. calibration estimation for 3D scene reconstruction for image sequences.

6. References

[BL03] Brown, M. and Lowe, D. G., “Recognising Panoramas”, Proc. of the 9th International Conference on Computer Vision (ICCV2003), pp. 1218-1225, Nice, France, 2003.

[BL02] Brown, M. and Lowe, D., “Invariant Features from Interest Point Groups”, British Machine Vision Conference, BMVC 2002, Cardiff, Wales, 2002.

[FS05] Foley, T. and Sugerman, J., ”KD-Tree Acceleration Structures for a GPU Raytracer”, Eurographics Report, Graphics Hardware, 2005.

[Fun05] Fung, J., “Computer Vision on the GPU”, GPU Gems 2, pp. 649-666, Addison-Wesley, 2005.

[GL04] Gordon, I. and Lowe, D.G., “Scene modeling, recognition and tracking with invariant image features”, International Symposium on Mixed and Augmented Reality (ISMAR), Arlington, USA, 2004.

[GLW*04] Govindaraju, N.K., Lloyd, B., Wang, W., Lin, M. and Manocha, D., „Fast Computation of Database Operations using Graphics Processors”, Proc. ACM SIGMOD 2004, Paris, France, 2004.

[GHLM05] Govindaraju, N.K., Henson, M., Lin, M. and Manocha, N., “Interactive Visibility Ordering of Geometric Primitives in Complex Environments”, Proc. ACM SIGGRAPH i3d, Washington DC, USA, 2005.

[HS88] Harris, C., Stephens, M., “A Combined Corner Edge Detector”, Proc. Alvey Vision Conference, pp. 189-192, Manchester, 1988.

[Har05] Harris, M., “Mapping Computational Concepts to GPUs”, GPU Gems 2, pp. 493-508, Addison-Wesley, 2005.

[KF05] Kilgariff, E., Fernando, R., “The GeForce 6 Series GPU Architecture”, GPU Gems 2, pp. 471-491, Addison-Wesley, 2005.

[Low04a] Lowe, D.G. 2004, “Object Recognition from local scale-invariant features”, Proc. International Conference on Computer Vision, pp.1150-1157, Corfu, Greece.

[Low04b] Lowe, D.G., “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, 2004.

[OLG*05] Owens, J.D., Luebke, D., Govindaraju, N.K., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J., “A Survey of General-Purpose Computation on Graphics Hardware”, Eurographics 2005, State of the Art Reports, Dublin, Ireland, 2005.

[Wit83] Witkin, A.P., “Scale-space filtering, Proc. International Joint Conference on Artificial Intelligence, pp. 1019-1022, Karlsruhe, Germany, 1983.

[Zel05] Zeller, C., “Introduction to the Hardware Graphics Pipeline”, ACM SIGGRAPH i3d, Invited Speech, Washington DC, USA, 2005.

 ISBN 978-80-86943-98-5