The 20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision

in co-operation with

EUROGRAPHICS

W S C G ' 2012

Conference Proceedings

Part I

Plzen

Czech Republic

June 26 - 28, 2012

Co-Chairs

Enhua Wu, University of Macau & Chinese Academy of Sciences, China
Vaclav Skala, University of West Bohemia, Czech Republic

Edited by
Vaclav Skala

Vaclav Skala – Union Agency

WSCG'2012 Conference Proceedings

Editor-in-Chief: Vaclav Skala
c/o University of West Bohemia, Univerzitni 8
CZ 306 14 Plzen
Czech Republic
http://www.VaclavSkala.eu

Managing Editor: Vaclav Skala

Published and printed by:
Vaclav Skala – Union Agency
Na Mazinách 9
CZ 322 00 Plzen
Czech Republic

Hardcopy: *ISBN 978-80-86943-79-4*

WSCG 2012

International Program Committee

Adzhiev,V. (United Kingdom)
Benes,B. (United States)
Bengtsson,E. (Sweden)
Benoit,C. (France)
Bilbao,J. (Spain)
Biri,V. (France)
Bittner,J. (Czech Republic)
Bouatouch,K. (France)
Bourke,P. (Australia)
Coquillart,S. (France)
Daniel,M. (France)
de Geus,K. (Brazil)
Debelov,V. (Russia)
Feito,F. (Spain)
Ferguson,S. (United Kingdom)
Flaquer,J. (Spain)
Gavrilova,M. (Canada)
Gudukbay,U. (Turkey)
Havran,V. (Czech Republic)
Hege,H. (Germany)
Chmielewski,L. (Poland)
Chover,M. (Spain)
Chrysanthou,Y. (Cyprus)
Jansen,F. (Netherlands)
Klosowski,J. (United States)
Magnor,M. (Germany)
Max,N. (United States)
Molla Vaya,R. (Spain)
Muller,H. (Germany)
Murtagh,F. (Ireland)
Myszkowski,K. (Germany)
Pan,R. (China)
Pasko,A. (United Kingdom)
Pedrini,H. (Brazil)
Platis,N. (Greece)
Rojas-Sola,J. (Spain)
Rokita,P. (Poland)
Rudomin,I. (Mexico)
Sakas,G. (Germany)
Santos,L. (Portugal)
Skala,V. (Czech Republic)
Slavik,P. (Czech Republic)
Sochor,J. (Czech Republic)
Sramek,M. (Austria)
Staadt,O. (Germany)
Stroud,I. (Switzerland)
Teschner,M. (Germany)
Tokuta,A. (United States)
Triantafyllidis,G. (Greece)
Vergeest,J. (Netherlands)
Vitulano,D. (Italy)
Weiss,G. (Germany)
Wu,E. (China)
Wuethrich,C. (Germany)
Zara,J. (Czech Republic)
Zemcik,P. (Czech Republic)
Zitova,B. (Czech Republic)

WSCG 2012

Board of Reviewers

Abad,F. (Spain)
Adzhiev,V. (United Kingdom)
Ariu,D. (Italy)
Assarsson,U. (Sweden)
Aveneau,L. (France)
Barthe,L. (France)
Battiato,S. (Italy)
Benes,B. (United States)
Benger,W. (United States)
Bengtsson,E. (Sweden)
Benoit,C. (France)
Beyer,J. (Saudi Arabia)
Biasotti,S. (Italy)
Bilbao,J. (Spain)
Biri,V. (France)
Bittner,J. (Czech Republic)
Bosch,C. (Spain)
Bouatouch,K. (France)
Bourdin,J. (France)
Bourke,P. (Australia)
Bruckner,S. (Austria)
Bruder,G. (Germany)
Bruni,V. (Italy)
Buriol,T. (Brazil)
Cakmak,H. (Germany)
Capek,M. (Czech Republic)
Cline,D. (United States)
Coquillart,S. (France)
Corcoran,A. (Ireland)
Cosker,D. (United Kingdom)
Daniel,M. (France)
Daniels,K. (United States)
de Geus,K. (Brazil)
De Paolis,L. (Italy)
Debelov,V. (Russia)
Dingliana,J. (Ireland)
Dokken,T. (Norway)
Drechsler,K. (Germany)
Durikovic,R. (Slovakia)
Eisemann,M. (Germany)
Erbacher,R. (United States)
Erleben,K. (Denmark)
Essert,C. (France)
Faudot,D. (France)
Feito,F. (Spain)
Ferguson,S. (United Kingdom)
Fernandes,A. (Portugal)
Flaquer,J. (Spain)
Flerackers,E. (Belgium)
Fuenfzig,C. (Germany)
Galo,M. (Brazil)
Garcia Hernandez,R. (Spain)
Garcia-Alonso,A. (Spain)
Gavrilova,M. (Canada)
Giannini,F. (Italy)
Gobron,S. (Switzerland)
Gonzalez,P. (Spain)
Gudukbay,U. (Turkey)
Guérin,E. (France)
Hall,P. (United Kingdom)
Hansford,D. (United States)
Haro,A. (United States)
Hasler,N. (Germany)
Hast,A. (Sweden)
Havran,V. (Czech Republic)
Hege,H. (Germany)
Hernandez,B. (Mexico)
Herout,A. (Czech Republic)
Hicks,Y. (United Kingdom)
Horain,P. (France)
House,D. (United States)
Chaine,R. (France)
Chaudhuri,D. (India)
Chmielewski,L. (Poland)
Choi,S. (Korea)
Chover,M. (Spain)
Chrysanthou,Y. (Cyprus)
Ihrke,I. (Germany)
Jansen,F. (Netherlands)
Jeschke,S. (Austria)
Jones,M. (United Kingdom)
Juettler,B. (Austria)
Kanai,T. (Japan)
Kim,H. (Korea)
Klosowski,J. (United States)
Kohout,J. (Czech Republic)
Krivanek,J. (Czech Republic)
Kurillo,G. (United States)
Kurt,M. (Turkey)
Lay Herrera,T. (Germany)
Lien,J. (United States)
Liu,S. (China)
Liu,D. (Taiwan)
Loscos,C. (France)
Lucas,L. (France)
Lutteroth,C. (New Zealand)
Maciel,A. (Brazil)
Madeiras Pereira,J. (Portugal)
Magnor,M. (Germany)
Manak,M. (Czech Republic)
Manzke,M. (Ireland)
Mas,A. (Spain)
Masia,B. (Spain)
Masood,S. (United States)
Matey,L. (Spain)
Matkovic,K. (Austria)
Max,N. (United States)
McDonnell,R. (Ireland)
McKisic,K. (United States)
Mestre,D. (France)
Molina Masso,J. (Spain)
Molla Vaya,R. (Spain)
Montrucchio,B. (Italy)
Muller,H. (Germany)

Murtagh,F. (Ireland)
Myszkowski,K. (Germany)
Niemann,H. (Germany)
Okabe,M. (Japan)
Oliveira Junior,P. (Brazil)
Oyarzun Laura,C. (Germany)
Pala,P. (Italy)
Pan,R. (China)
Papaioannou,G. (Greece)
Paquette,E. (Canada)
Pasko,A. (United Kingdom)
Pasko,G. (United Kingdom)
Pastor,L. (Spain)
Patane,G. (Italy)
Patow,G. (Spain)
Pedrini,H. (Brazil)
Peters,J. (United States)
Peytavie,A. (France)
Pina,J. (Spain)
Platis,N. (Greece)
Plemenos,D. (France)
Poulin,P. (Canada)
Puig,A. (Spain)
Reisner-Kollmann,I. (Austria)
Renaud,c. (France)
Reshetov,A. (United States)
Richardson,J. (United States)
Rojas-Sola,J. (Spain)
Rokita,P. (Poland)
Rudomin,I. (Mexico)
Runde,C. (Germany)
Sacco,M. (Italy)
Sadlo,F. (Germany)
Sakas,G. (Germany)
Salvetti,O. (Italy)
Sanna,A. (Italy)
Santos,L. (Portugal)
Sapidis,N. (Greece)
Savchenko,V. (Japan)
Sellent,A. (Germany)
Sheng,B. (China)
Sherstyuk,A. (United States)
Shesh,A. (United States)
Schultz,T. (Germany)
Sirakov,N. (United States)
Skala,V. (Czech Republic)
Slavik,P. (Czech Republic)
Sochor,J. (Czech Republic)
Solis,A. (Mexico)
Sourin,A. (Singapore)
Sousa,A. (Portugal)
Sramek,M. (Austria)
Staadt,O. ()
Stroud,I. (Switzerland)
Subsol,G. (France)
Sunar,M. (Malaysia)
Sundstedt,V. (Sweden)
Svoboda,T. (Czech Republic)
Szecsi,L. (Hungary)
Takala,T. (Finland)
Tang,M. (China)
Tavares,J. (Portugal)
Teschner,M. (Germany)
Theussl,T. (Saudi Arabia)
Tian,F. (United Kingdom)
Tokuta,A. (United States)
Torrens,F. (Spain)
Triantafyllidis,G. (Greece)
TYTKOWSKI,K. (Poland)
Umlauf,G. (Germany)
Vavilin,A. (Korea)
Vazquez,P. (Spain)
Vergeest,J. (Netherlands)
Vitulano,D. (Italy)
Vosinakis,S. (Greece)
Walczak,K. (Poland)
WAN,L. (China)
Wang,C. (Hong Kong SAR)
Weber,A. (Germany)
Weiss,G. (Germany)
Wu,E. (China)
Wuensche,B. (New Zealand)
Wuethrich,C. (Germany)
Xin,S. (Singapore)
Xu,D. (United States)
Yang,X. (China)
Yoshizawa,S. (Japan)
YU,Q. (United Kingdom)
Yue,Y. (Japan)
Zara,J. (Czech Republic)
Zemcik,P. (Czech Republic)
Zhang,X. (Korea)
Zhang,X. (China)
Zillich,M. (Austria)
Zitova,B. (Czech Republic)
Zwettler,G. (Austria)

WSCG 2012

Communications Proceedings

Contents

A Bigger Mathematical Picture for Computer Graphics

Eric Lengyel
Terathon Software LLC
http://www.terathon.com/lengyel/
USA

ABSTRACT

Some of the most brilliant mathematical discoveries of the 1800s were pushed aside for over a century in favor of the vector analysis and linear algebra that we are all familiar with. However, these old ideas have recently been rediscovered in the field of computer graphics by researchers who understand how they can unify many of the geometric operations that are used every day.

This talk introduces the basic concepts of the exterior algebra and presents a bigger mathematical picture that enables a deeper understanding of the homogeneous representation of points, lines, and planes, as well as the operations that can be performed among them using the progressive and regressive products. Some emphasis is placed on the history of related mathematics and the past development of incomplete pieces of the bigger picture, such as Plücker coordinates. The goal is to help the audience unlearn some longstanding misnomers in 3D geometry and to provide the knowledge of a larger, unified world into which many familiar mathematical concepts fit together.

BRIEF BIOGRAPHY

Eric Lengyel is the founder of Terathon Software and the creator of the C4 Engine, a comprehensive technology platform for games and virtual simulations. He holds a Ph.D. in Computer Science from the University of California at Davis and a Masters Degree in Mathematics from Virginia Tech.

Eric is the best-selling author of the book *Mathematics for 3D Game Programming & Computer Graphics*, and he is the series editor for the new *Game Engine Gems* series. Eric is also a member of the editorial board for the *Journal of Graphics Tools*, and he is a major contributor to the successful *Game Programming Gems* series.

Eric previously worked in the advanced technology group at Naughty Dog where he developed the driver architecture for the Playstation 3. Prior to that, was the lead programmer for Sierra Studio's popular adventure game *Quest for Glory V*, and he worked on OpenGL in Apple's graphics and imaging department.

Overcoming Physical Limitations of Display Devices in Rendering

Karol Myszkowski
Max-Planck-Institut für Informatik
http://www.mpi-inf.mpg.de/
Germany

ABSTRACT

The knowledge of human visual system (HVS) enables more efficient image rendering by overcoming physical constraints of display devices. This talk presents a number of successful examples of embedding HVS models into real-time rendering pipelines. In particular, I discuss the problem of improving the appearance of highlights and light sources by boosting their apparent brightness using the temporal glare technique. Also, I discuss how to overcome physical contrast limitations of display devices by using the 3D unsharp masking technique to boost the apparent contrast. Also, I present techniques for apparent resolution enhancement, which enable showing image details beyond the physical pixel resolution of the display device. Finally, I discuss the role of perception in context of stereovision and accommodation/vergence conflict reduction

BRIEF BIOGRAPHY

Karol Myszkowski is a tenured senior researcher at the MPI Informatik, Saarbruecken, Germany. In the period from 1993 till 2000 he served as an associate professor in the Department of Computer Software at the University of Aizu, Japan. In the period from 1986 till 1992 he worked for Integra, Inc. a Japan-based, company specialized in developing rendering and global illumination software. He received his PhD (1991) and habilitation (2001) degrees in computer science from Warsaw University of Technology (Poland). In 2011 he was awarded with a lifetime professor title by the President of Poland. His research interests include perception issues in graphics, high dynamic range imaging, global illumination and rendering. Karol published and lectured on these topics widely including ACM Siggraph/Siggraph Asia Courses in 2001, 2002, 2004, 2006, and 2012. He also co-chaired Rendering Symposium in 2001, ACM Symposium on Applied Perception in Graphics and Visualization in 2008, Spring Conference on Computer Graphics 2008, and Graphicon 2012

Physically Based Weathering Simulation of Natural Objects Based on Biological Analysis

Enhua Wu
State Key Lab. of Computer Science, Chinese Academy of Sciences, Beijing
&
University of Macau, Macao，China

ABSTRACT

The Weathering effect of nature objects or natural scenes is a common phenomenon in our daily life. However, little investigation has been made to the phenomenon so far in computer graphics field. The weathering procedure on the nature objects such as plants, trees, grasses etc. is a slowly changing process, and in fact it is involved with a comprehensive drying procedure made towards the biological structure of the nature objects, in terms of the shape change of the objects & the color change of their appearance. With regard to the shape change or deformation, a physically based mechanical calculation is applied to the biological components incurred by the drying effect in our solution. On the other hand, the change of color appearance could be simulated based on the synthesis to the color spectrum of the samples collected in the weathering process of the objects. The simulation based the scheme will be demonstrated by the simulation result to the trees, grassland, fruits etc.

BRIEF BIOGRAPHY

Dr. Enhua Wu completed his BSc in Tsinghua University, Beijing in 1970 and received his Ph.D degree from Dept. of Computer Science, University of Manchester, England in 1984. Since 1985 he has been working at the Institute of Software, Chinese Academy of Sciences, as a director of the Research Dept. of Fundamental Theory and Advanced Technology until 1998. Since September of 1997, he has been also invited as a full professor of University of Macau (UM).

Dr. Wu's main interests are Realistic Image Synthesis, Virtual Reality and Scientific Visualization. Now he is an Associate Editor-in-Chief of the Journal of Computer Science and Technology (Science Press and Springer) and the editorial board member of TVC, CAVW, IJIG, IJVR, IJSI. He has been also in recent years invited as a keynote speaker or chairing works in a number of international conferences such as ACM VRST2010, CASA2011, ACM VRCAI2008-2012, IEEE VR2011-12 etc.

A Beginners Guide to Dual-Quaternions

What They Are, How They Work, and How to Use Them for 3D Character Hierarchies

Ben Kenwright
School of Computing Science, Newcastle University
Newcastle Upon Tyne, United Kingdom
b.kenwright@ncl.ac.uk

ABSTRACT

In this paper, we give a beginners guide to the practicality of using dual-quaternions to represent the rotations and translations in character-based hierarchies. Quaternions have proven themselves in many fields of science and computing as providing an unambiguous, un-cumbersome, computationally efficient method of representing rotational information. We hope after reading this paper the reader will take a similar view on dual-quaternions. We explain how dual number theory can extend quaternions to dual-quaternions and how we can use them to represent rigid transforms (i.e., translations and rotations). Through a set of examples, we demonstrate exactly how dual-quaternions relate rotations and translations and compare them with traditional Euler's angles in combination with Matrix concatenation. We give a clear-cut, step-by-step introduction to dual-quaternions, which is followed by a no-nonsense how-to approach on employing them in code. The reader, I believe, after reading this paper should be able to see how dual-quaternions can offer a straightforward solution of representing rigid transforms (e.g., in complex character hierarchies). We show how dual-quaternions propose a novel alternative to pure Euler-Matrix methods and how a hybrid system in combination with matrices results in a faster more reliable solution. We focus on demonstrating the enormous rewards of using dual-quaternions for rigid transforms and in particular their application in complex 3D character hierarchies.

Keywords

Dual-Quaternion, 3D, Real-Time, Character Hierarchies, Rigid Transformation

1. INTRODUCTION

Real-time dynamic 3D character systems combine key framed animations, inverse kinematics (IK) and physics-based models to produce controllable, responsive, realistic motions. The majority of character-based systems use a skeleton hierarchical composition of rigid transforms. Each rigid transform has six degrees of freedom (DOF) that consists of three translational and three rotational components. Matrices are the most popular method of storing and combining these transforms. While matrices are adequate, we ask the question, is there a better method? In this paper, we address the advantages and disadvantages of matrices while proposing a novel alternative based on quaternions called dual-quaternions. The purpose of this paper is to present a beginner's guide to dual-quaternions in sufficient detail that the reader can begin to use them as a practical problem-solving tool for rigid character transforms. This paper covers the basics of dual-quaternions and their application to complex hierarchical systems with many DOF.

Dual-quaternions are interesting and important because they cut down the volume of algebra. They make the solution more straightforward and robust. *They allow us to unify the translation and rotation into a single state*; instead of having to define separate vectors. While matrices offer a comparable alternative to dual-quaternions, we argue that they can be inefficient and cumbersome in comparison. In fact, dual-quaternions give us a compact, un-ambiguous, singularity-free, and computational minimalistic rigid transform. In addition, dual-quaternions have been shown to be the most efficient and most compact form of representing rotation and translation. Dual-quaternions can easily take the place of matrices in hierarchies at no additional cost. For rigid transform hierarchies that combine and compare rigid transforms on a frame-by-frame bases (e.g., character inverse kinematics (IK) and joint constraints), alternative methods such as matrices need to be converted to quaternions to generate reliable contrast data; this can be done without any conversion using dual-quaternions.

Many students have a great deal of trouble understanding essentially what quaternions are and how they can represent rotation. So when the subject of dual-quaternions is presented, it is usually not welcomed with open arms. Dual-quaternions are a break from the norm (i.e., matrices) which we hope to entice the reader into embracing to represent their

rigid transforms. The reader should walk away from this paper with a clear understanding of what dual-quaternions are and how they can be used.

The majority of computer scientists are familiar with vectors, matrices, and quaternions. They provide a set of tools to help solve problems. This paper presents the case for adding dual-quaternions to this set of tools.

The contribution of this paper is the explanation and demonstration of dual-quaternions in a sufficiently detailed way that the reader can begin to appreciate their practical problem-solving advantages. We use character-based hierarchies as a base method to illustrate the realistic reward of dual-quaternions in time critical systems (e.g., games).

This paper presents dual-quaternions as a method for representing rigid transforms in complex character hierarchies with a large number of DOF. We explain how to implement a basic dual-quaternion class and combine dual-quaternions through straightforward multiplication to work in place of matrices.

The roadmap for the rest of the paper is as follows: we begin with a review of recent and related work that emphasises the power of dual-quaternions. We review familiar rigid transform methods and their advantages and disadvantages. We then outline the primary reasons for using dual-quaternions and why you would want to use them for rigid transforms over other methods. We then give the background mathematical information for dual numbers, quaternions and dual-quaternions. The following sections then focus on the practical aspects of dual-quaternions. We discuss a variety of experiments with computer simulations and character hierarchies in relation to dual-quaternion. Finally, the end section presents the conclusion and proposed future work.

2. RELATED WORK

The dual-quaternion has been around since 1882 [CLIF82] but has gained less attention compared to quaternions alone. Comparable to quaternions the dual-quaternions have had a taboo associated with them, whereby students avoid quaternion and hence dual-quaternions. While the robotics community has started to adopt dual-quaternions in recent years, the computer graphics community has not embraced them as whole-heartedly. We review some recent work which has taken hold and has demonstrated the practicality of dual-quaternions, both in robotics and computer graphics.

2.1. Computer Graphics

Kavan [KCŽO08] demonstrated the advantages of dual-quaternions in character skinning and blending.

Ivo [IVIV11] extended Kavans [KCŽO08] work with dual-quaternions and qtangents as an alternative method for representing rigid transforms instead of matrices, and gives evidence that the results can be faster with accumulated transformations of joints if the inferences per vertex are large enough.

Selig [SELI11] address the key problem in computer games. Examining the problem of solving the equations of motion in real-time and puts forward how dual-quaternion give a very neat and succinct way of represent rigid-body transformations.

Vasilakis [VAFU09] discussed skeleton-based rigid-skinning for character animation.

Kuang [KMLX11] presented a strategy for creating real-time animation of clothed body movement.

2.2. Robotics

Pham [PPAF10] solved linked chain inverse kinematic (IK) problems using Jacobian matrix in the dual-quaternion space.

Malte [SCHI11] used a mean of multiple computational (MMC) model with dual-quaternions to model bodies.

Ge [GVMC98] demonstrated dual-quaternions to be an efficient and practical method for interpolating three-dimensional motions.

Yang-Hsing [LIWC10] calculated the relative orientation using dual-quaternions.

Perez [PEMC04] formulated dynamic constraints for articulated robotic systems using dual-quaternions.

3. FAMILIAR PHYSICAL CONCEPTS

We review the most common methods of representing rigid body orientations and translations in our physical world (three spatial dimensions). While orientation and rotation are familiar concepts, there are many ways to represent them both mathematically and computationally, each with their own strengths and weaknesses. We briefly describe four of the most popular methods of representing rigid transforms in character systems. This helps illustrate the mathematical and computational issues that occur. The four alternate methods we compare mathematically and computationally to dual-quaternions are:

- Matrices
- Axis-Angles
- Euler-Angles
- Quaternions

} + Translation

Each alternative method needs to represent both the orientation and translation. In some cases this is achieved by using two separate state variables and combining them separately, while matrices and dual-quaternions give us a unified state variable.

For each case we focus on issues of interpolation, computational speed, mathematical robustness and distance metrics.

The properties we look for to represent the rigid body transform are:

Robustness – be continuous and not contain any discontinuities (such as gimbal lock with Euler's angles which we discuss later). Contain a unique representation, where some methods contain redundant information, such that several or an infinite number of elements can represent the same transform.

Efficiency – should consume the smallest necessary amount of space and be computationally fast. We would like a minimum number of calculations to combine and convert between alternative representations (e.g., cost to convert between matrices and Euler angles).

Ease of Use – can be used without too many complications.

3.1. Orientation and Translation

It might seem intuitive how objects are rotated and translated. For example, we can pick up any object around us and spin (rotate) and translate (move) it without thinking. However, how do we model this computationally and mathematically? The following sub-sections are devoted to the explanation and understanding of these basic principles.

For methods which are formed from separate orientation and translational information, we can analyse their workings by considering orientation and translation separately and combining them at the end of each transform.

3.2. Translation

The translation coordinates are relatively simple to work with. They compose of the scalar values along each of the principle axes (x, y and z). The computed orientations are combined with the translations by rotating the principle axis.

3.3. Euler-Angles

A familiar way of representing the orientation and translation in character systems is to factor it into three sequential angles around the principle orthogonal axes (x, y and z).

Euler's angles in 3D <u>do not</u> (in-general) *commute* under composition.

In practice, the angles are used by inserting them into matrices. The product of the three angle-matrices produces the Euler angle set. There are twelve possible products: XYZ, XYX, YZX, YZY, ZXY, ZXZ, XZY, XZX, YXZ, YXY, ZYX, and ZYZ. These are the order the rotations are applied in. For example, the factorization XYZ, would mean rotate round X then Y then Z.

To work with Euler angles we convert them to matrices:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$$

$$\mathbf{Z} = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Combining the translation is just a matter of rotating the translational components (x, y and z) by the rotation.

To combine and calculate interpolating differences requires us to find the equivalent axis-angle of the two orientations and extrapolate the Euler angles.

- Create a matrix for each Euler angle.
- Multiply the three matrices together.
- Extract axis-angle from resulting matrix.

Converting, combining, and extracting Euler angles is computationally expensive. Moreover, Euler angles can have discontinuities around 0 and 2π, since the components live on separate circles and not a single vector space.

3.3.1. Advantages

People prefer Euler angles as they can comprehend them effortlessly and can create orientations with them without difficulty. They are also very intuitive and have a long history in physics and graphics and can make certain integrals over rotational space easier.

Euler angles are minimalistic and require only three parameters; however, we show later how four parameters are better than three. Furthermore, since the angles are used directly, there is no drifting or the need for normalization.

3.3.2. Disadvantages

Euler angles suffer from singularities - angles will instantaneously change by up to 2π radians as other angles go through the singularity; Euler angles are virtually impossible to use for sequential rotations. There are twelve different possible Euler angle rotation sequences - XYZ, XYX, XZY, and so on. There is no one "simplest" or "right" set of Euler angles. To derive a set of Euler angles you must know which rotational sequence you are using and stick to it.

In practice when Euler angles are needed; the underlying rotation operations are done using quaternions and are converted to Euler angles for the task at hand.

3.3.3. Gimbals Lock

The coordination singularity in Euler's angles is commonly referred to as gimbals lock. A gimbal is a physical device consisting of spherical concentric hoops with pivots connecting adjacent hoops, allowing them to rotate within each other (see Figure 1).

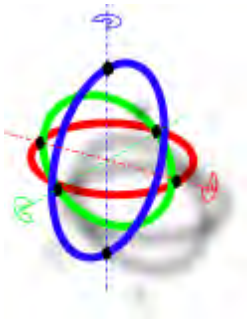


Figure 1. Gimbal with points of rotation indicated.

A gimbal is constructed by aligning three rings and attaching them orthogonally. Gimbals are often seen in gyroscopes used by the aeronautical industry.

As objects are rotated, they approach gimbal lock the singularity will cause numerical ill-conditioning, often evidented physically by the gimbal wiggling madly around as it operates near the singularity. This is one reason why the aerospace industry, early on, switched to quaternions to represent orientation – satellites, rockets and airplanes would have their navigation gyro lock up and could cause them to crash.

3.3.4. Interpolation

The major problem with Euler interpolation is that they have problems while interpolating near gimbals lock regions. When close to a gimbal lock singularity the interpolation become jittery and noise ridden; eventually becoming random and unstable as it converges on the singularity.

If Euler angles are interpolated linearly the resulting path will not take the shortest path between the endpoints as it does in vector space [ALMA92].

3.4. Axis-Angle

The axis-angle is represented by a unit axis and angle $(\hat{n}, \theta)$ pair. This axis-angle representation can easily be converted to and from a matrix.

It is difficult to combine the axis-angle elements in their native form; usually being converted to an alternate representation for concatenation (e.g., matrices, quaternions).

3.4.1. Advantages

The greatest single advantage of the axis-angle representation is that it directly represents the action of rotation, while being straightforward and intuitive to work with.

3.4.2. Disadvantages

We can renormalize the axis since it is a unit vector, but numerical errors can still creep into the angle portion.

Infinite number of angle choices (multiples of 2π), so two axis-angle pairs can still refer to the same rotation but be different.

Axis-angle interpolation cannot be done using linear interpolation of the four elements. Interpolating between the four elements naively in this way does not give the shortest path.

Interpolating the angle alone can introduce discontinuities as the angle crosses from 0 to 2π. These 'jumps' are highly undesirable and can cause anarchy with the interpolation and numerical integration schemes.

3.5. Matrices

Representing a rigid transform using a matrix we extend a 3x3 rotation matrix to include translation information which makes it a 4x3 matrix. While a 4x3 matrix is the most efficient, on most occasions a 4x4 matrix is used because of availability.

The 3x3 part of the matrix consists of three orthogonal column vectors which are of unit magnitude.

A transform matrix can transform a vector coordinate by simply matrix multiplication:

$$\mathbf{y} = \mathbf{T}\,\mathbf{x}$$

where T is a transform matrix, x a vector coordinate and y the transformed result.

If the position and basis vectors are known, the transform matrix can trivially be produced, because each of the columns in the 3x3 part of the matrix represent the base vectors and the bottom row the translation.

The combination of matrix elements is achieved through simple multiplication. Matrices are not commutative and therefore their matrix representation of rigid body transforms is non-commutative as well.

3.5.1. Advantages

Matrices are taught in linear algebra early on in colleges so this makes them more familiar and favourable. In addition, a great many algorithms have been formulated and tested with matrices and so people choose them instinctively first.

3.5.2. Disadvantages

While matrices might seem to be the utopia, they in-fact can be found to have several problems.

Firstly, they take a minimum of 12 parameters to represent a structure with only six DOF; if memory is at a premium this can be undesirable.

Secondly, the rotational part of the matrix is composed of orthogonal columns which can drift and introduce unwanted scaling and sheering. We can re-normalize the matrix using Gram-Schmidt method [GILB86] but this can be computationally expensive.

Thirdly, interpolating between matrices is difficult. The three columns forming the orthogonal axis directions in the rotation part of the matrix do not represent the vector space and cannot be interpolated.

Finally, it is difficult to visualize a matrix and the axis-angle component about which it will rotate and translate.

3.6. Method Summary

We have outlined and examined current methods for representing a robust, practical and viable hierarchical rigid body solution. We now follow on from this by introducing and explaining how and why dual-quaternions stand-out above these methods.

4. WHY DUAL-QUATERNIONS?

We use dual-quaternions as a tool for expressing and analyzing the physical properties of rigid bodies. Dual-quaternions can formulate a problem more concisely, solve it more rapidly and in fewer steps, present the result more plainly to others, be put into practice with fewer lines of code and debugged effortlessly. Furthermore, there is no loss of efficiency; dual-quaternions can be just as efficient if not more efficient than using matrix methods. In all, there are several reasons for using dual-quaternions, which we summarize:

- Singularity-free
- Un-ambiguous
- Shortest path interpolation
- Most efficient and compact form for representing rigid transforms [SCHI11] - (3x4 matrix 12 floats compared to a dual-quaternion 8 floats)
- Unified representation of translation and rotation
- Can be integrated into a current system with little coding effort
- The individual translation and rotational information is combined to produce a single invariant coordinate frame [GVMC98]

5. DUAL NUMBERS

Clifford [CLIF82] introduced dual numbers; similar to complex numbers that consists of two parts known as the real and complex component. Dual numbers break the problem into two components and are defined as:

$$z = r + d\varepsilon \text{ with } \varepsilon^2 = 0 \text{ but } \varepsilon \neq 0$$

where ε is the dual operator, r is the real part and d the dual part. Similar to complex number theory, where i is added to distinguish the real and complex components, the dual operator ε is used in the same way.

The dual number theory can be extended to other concepts, such as vectors and real numbers, but we focus on their applicability in conjunction with quaternions to represent rotation and translation transforms.

5.1. Dual Number Arithmetic Operations

Dual numbers can perform the fundamental arithmetic operations below:

Addition

$$(r_A + d_A\varepsilon) + (r_B + d_B\varepsilon) = (r_A + r_B) + (d_A + d_B)\varepsilon$$

Multiplication

$$\begin{aligned}(r_A + d_A\varepsilon)(r_B + d_B\varepsilon) &= r_A r_B + r_A d_B\varepsilon + r_B d_A\varepsilon + d_A d_B\varepsilon^2 \\ &= r_A r_B + (r_A d_B + r_B d_A)\varepsilon\end{aligned}$$

Division

$$\begin{aligned}\frac{(r_A + d_A\varepsilon)}{(r_B + d_B\varepsilon)} &= \frac{(r_A + d_A\varepsilon)}{(r_B + d_B\varepsilon)}\frac{(r_B - d_B\varepsilon)}{(r_B - d_B\varepsilon)} \\ &= \frac{r_A r_B + (r_B d_A - r_A d_B)\varepsilon}{(r_B)^2} \\ &= \frac{r_A r_B}{r_B^2} + \frac{r_B d_A - r_A d_B}{r_B^2}\varepsilon\end{aligned}$$

Further reading on the subject of dual numbers is presented by Gino [BERG09].

5.2. Dual Number Differentiation

Dual numbers differentiate in the same way as any other vector using elementary calculus principles, e.g.:

$$\frac{\mathrm{d}}{\mathrm{dx}}\mathbf{s}(x) = \lim_{\delta x \to 0}\frac{\mathbf{s}(x + \delta x) - \mathbf{s}(x)}{\delta x}$$

The derivative of a dual number is another dual number. Remarkably, the dual operator's condition $\varepsilon^2 = 0$ enables us to take advantage of Taylor series to find the differentiable. Where we can see below, if we substituting a dual number into Taylor series, we get:

$$\begin{aligned}f(r_A + d_A\varepsilon) &= f(r_A) + \frac{f'(r_A)}{1!}d_A\varepsilon + \frac{f''(r_A)}{2!}(d_A\varepsilon)^2 + \frac{f'''(r_A)}{3!}(d_A\varepsilon)^3 + \ldots \\ &= f(r_A) + \frac{f'(r_A)}{1!}d_A\varepsilon + 0 + 0 + \ldots \qquad (as,\ \varepsilon^2 = 0) \\ &= f(r_A) + f'(r_A)d_A\varepsilon\end{aligned}$$

Remarkably, the Taylor series result gives us an exceptionally tidy answer; from this we use dual number arithmetic and substitution to find the solution to any differential.

The derivative also enables us to find the tangent of an arbitrary point p on a given parametric curve that is equal to the normalized dual part of the point p.

6. QUATERNIONS

Quaternions were introduced by Hamilton in 1866 [HAMI86] and have had a rollercoaster of a time with acceptance. Quaternions are an extension of complex number-theory to formulate a four dimensional manifold. A quaternion is defined as:

$$\mathbf{q} = w + (x\mathbf{i} + y\mathbf{j} + z\mathbf{k})$$

where *w*, *x*, *y* and *z* are the numerical values, while *i*, *j* and *k* are the imaginary components.

The imaginary components properties:

$$i^2 = j^2 = k^2 = -1$$

and

$$\begin{aligned} ij &= k, & ji &= -k \\ jk &= i, & kj &= -i \\ ki &= j, & ik &= -j \end{aligned}$$

It is more common to represent the quaternion as two components, the vector component (x, y and z) and the scalar component (w).

$$\mathbf{q} = (w, \mathbf{v})$$

For further reading on the workings of quaternions and their advantages I highly recommend reading McDonalds [MCDO10] introductory paper for students.

6.1. Quaternion Arithmetic Operations

Since we are combining quaternions with dual number theory, we give the elementary quaternion arithmetic operations below:

Scalar Multiplication

$$s\mathbf{q} = (sw, s\mathbf{v})$$

where *s* is a scalar value.

Addition

$$\mathbf{q}_1 + \mathbf{q}_2 = (w_1 + w_2,\ \mathbf{v}_1 + \mathbf{v}_2)$$

Multiplication

$$\mathbf{q}_1\mathbf{q}_2 = (w_1w_2 - v_1v_2,\ \ w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + (\mathbf{v}_1 \times \mathbf{v}_2))$$

Conjugate

$$\mathbf{q}^* = (w, -\mathbf{v})$$

Magnitude

$$\| \mathbf{q} \| = \mathbf{q}\mathbf{q}^*$$

For a unit quaternion, $\| \mathbf{q} \| = 1$. The unit quaternion is used to represent a rotation of an angle θ, radians about a unit axis $\mathbf{n}$, in three-dimensional space:

$$\mathbf{q} = (\cos(\frac{\theta}{2}),\ \mathbf{n}\sin(\frac{\theta}{2}))$$

6.2. Quaternion Interpolation

An extremely important quality of quaternions that make them indispensable in animation systems is their ability to interpolate two or more quaternions smoothly and continuously. Shoemake [SHOE85], presents an outstanding paper on using quaternion curves for animating rotations. Furthermore, it should be noted, the spherical linear interpolation (SLERP) properties of quaternions are inherited by dual-quaternions.

7. DUAL-QUATERNIONS

When quaternions are combined with dual number theory, we get dual-quaternions which was presented by Clifford in 1882 [CLIF82]. While the unit quaternion only has the ability to represent rotation, the unit dual-quaternion can represent both translation and rotation. Each dual-quaternion consists of eight elements or two quaternions. The two quaternion elements are called the real part and the dual part.

$$\mathbf{q} = \mathbf{q}_r + \mathbf{q}_d\varepsilon$$

where $\mathbf{q}_r$ and $\mathbf{q}_d$ are quaternions. Combining the algebra operations associated with quaternions with the additional dual number ε, we can form the dual-quaternion arithmetic.

7.1. Dual-Quaternion Arithmetic Operations

The elementary arithmetic operations necessary for us to use dual-quaternions are:

Scalar Multiplication

$$s\mathbf{q} = s\mathbf{q}_r + s\mathbf{q}_d\varepsilon$$

Addition

$$\mathbf{q}_1 + \mathbf{q}_2 = \mathbf{q}_{r1} + \mathbf{q}_{r2} + (\mathbf{q}_{d1} + \mathbf{q}_{d2})\varepsilon$$

Multiplication

$$\mathbf{q}_1\mathbf{q}_2 = \mathbf{q}_{r1}\mathbf{q}_{r2} + (\mathbf{q}_{r1}\mathbf{q}_{d2} + \mathbf{q}_{d1}\mathbf{q}_{r2})\varepsilon$$

Conjugate

$$\mathbf{q}^* = \mathbf{q}_r^* + \mathbf{q}_d^*\varepsilon$$

Magnitude

$$\| \mathbf{q} \| = \mathbf{q}\mathbf{q}^*$$

Unit Condition

$$\| \mathbf{q} \| = 1$$

$$\mathbf{q}_r^*\mathbf{q}_d + \mathbf{q}_d^*\mathbf{q}_r = 0$$

The unit dual-quaternion is our key concern as it can represent any rigid rotational and translational transformations.

The rigid rotational and translational information for the unit dual-quaternion is:

$$\mathbf{q}_r = \mathbf{r}$$

$$\mathbf{q}_d = \frac{1}{2}\,\mathbf{t}\,\mathbf{r}$$

where $\mathbf{r}$ is a unit quaternion representing the rotation and $\mathbf{t}$ is the quaternion describing the translation represented by the vector $\mathbf{t} = (0, ,, .$

The dual-quaternion can represent a pure rotation the same as a quaternion by setting the dual part to zero.

$$\mathbf{q}_r = [\cos(\frac{\theta}{2}),\ \mathbf{n}_x \sin(\frac{\theta}{2}),\ \mathbf{n}_y \sin(\frac{\theta}{2}),\ \mathbf{n}_z \sin(\frac{\theta}{2})\,][0,0,0,0]$$

To represent a pure translation with no rotation, the real part can be set to an identity and the dual part represents the translation.

$$\mathbf{q}_t = [1,0,0,0][0, \frac{\mathbf{t}_x}{2}, \frac{\mathbf{t}_y}{2}, \frac{\mathbf{t}_z}{2}]$$

Combining the rotational and translational transforms into a single unit quaternion to represent a rotation followed by a translation we get:

$$\mathbf{q} = \mathbf{q}_t \times \mathbf{q}_r$$

This arithmetic operation defines how we transform a point *p*, using a unit dual-quaternion:

$$\mathbf{p}' = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^*$$

where $\mathbf{q}$ and $\mathbf{q}^*$ represent a dual-quaternion transform and its conjugate; while $\mathbf{p}$ and $\mathbf{p}'$ represent our point inserted into a quaternion and its resulting transform.

8. PORTING EXISTING CODE TO DUAL-QUATERNIONS

A dual-quaternion consists of two quaternions, but is represented by a single variable Q. Systems that have been constructed using separate translation and rotation (vector for translation and quaternion for rotation) in combination with matrices schemes are easily modified to use dual-quaternions for spatial information.

1. For each link, construct a dual-quaternion Q from the rotation and translation information.
2. Real part of the quaternion is the rotation quaternion *r*. The dual part is calculated by multiplying the quaternion *r* and translation component *t*, e.g.:

$$Qr = r$$

$$Qd = 0.5\ (0,t)\ r$$

3. Combine transformations as you would matrices using multiplication.
4. If necessary, for long chains, the dual-quaternion should be re-normalized (to mend drift and maintain a unit dual-quaternion).
5. To get the homogeneous transformation matrix, convert the dual-quaternion by extracting the translational and rotational components.
6. The extracted rotation quaternion r and vector translation information is extracted using:

$$r = Qr$$

$$t = 2\ Qd\ Qr^*$$

Dual-quaternion multiplication is more efficient than matrix multiplication and can effortlessly be converted back to a matrix when needed. Dual-quaternions, unlike Euler angles, do not present issues like "*gimbal lock*" and hence, are ideal for complex articulated hierarchies.

9. COMPLEX CHARACTER HIERARCHY FORWARD KINEMATICS

The focus of our attention is with rigid hierarchies having a large number of DOF. Humans have a tremendous amount of flexibility which we emulate and analyze using numerical and mathematical models. Forward kinematics is the method of concatenating local positions and rotations together to give their global ones. The forward kinematic method for concatenating transforms is the same for dual-quaternions and matrices; which use simple multiplication to propagate transforms between the connected links.

For example, the concatenation of transforms with matrices and dual-quaternions:

Matrix

$$\mathbf{M}_{03} = \mathbf{M}_0\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3$$

Dual-Quaternion

$$\mathbf{q}_{03} = \mathbf{q}_0\mathbf{q}_1\mathbf{q}_2\mathbf{q}_3$$

where the subscript represents the transform, matrix transform $\mathbf{M}_0$ corresponding to dual-quaternion transform $\mathbf{q}_0$.

10. EXPERIMENTAL RESULTS

We used traditional matrix methods during initial character transformation experiments; e.g., inverse kinematic (IK) and animation blending to demonstrate their numerous problems. Matrix methods are a popular choice and solutions to these problems have been developed; we used some of these engineering solutions. Of course, these workarounds to these problems introduced an additional computational cost. Furthermore, certain circumventions to overcome a problem often introduced errors in other areas. One such engineering solution for reducing the impact of drift

and concatenation error was to renormalize the matrices at each level (and at each update frame). The error reduced skewing and scaling but manifested itself in the ideal global end-link orientations and positions being inaccurate.

To demonstrate the problems, we constructed numerous test cases to emphasis them. We also demonstrate and explore how dual-quaternions can represent rigid body character based systems.

10.1. Rigid Body Transform Chains

We constructed a straightforward IK solver that would follow a target end-effector. To mimic how a character would move his arm or leg. The end-effector had six DOF, which the IK solver had to work with to meet its target goal.

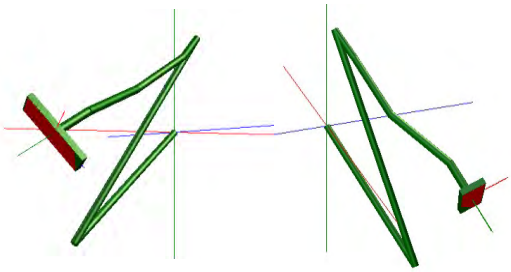


Figure 2. Rigid body links attached in a single hieararchy frame. (Draw ideal(red) and calculated end-effector (green).

The hierarchy is composed of rigid links. Each link held a rotation and translation in the form of a matrix or dual-quaternion. For calculations, the axis-angle and translation could be extracted and used when needed. Local transforms were combined from the root to the end-effectors. Concatenation of the transforms throughout the levels was achieved by multiplying parent transforms with current transforms.

Certain orientation and translation configurations produced errors in the output, shown in Figure 3. These errors presented themselves as skewing and scaling manifestations.

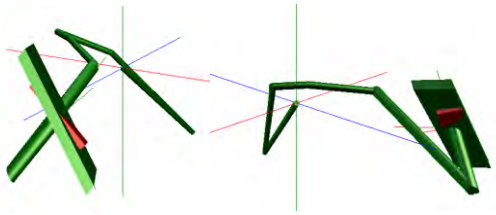


Figure 3. Artifact error when matrices representing translation and orientation in linked hierarchies.

Early workarounds to amend the problem were to repair the matrix at each level in the hierarchy by ortho-normalizing the rotational component. While ortho-normalizing the matrix reduced scaling and skewing artifacts, alternative errors manifest themselves in alternative forms.

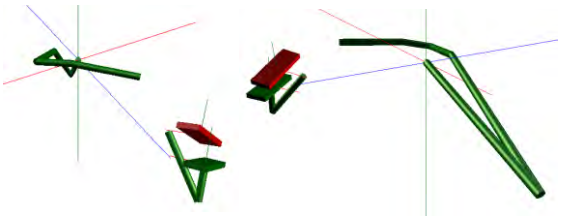


Figure 4. Ortho-normalizing matrices in hierarchies in an attempt to reduce errors.

Ortho-normalizing the rotational part of the transform matrix between updates removed scaling and skewing problems. The joints presented discontinuity errors in the frames hierarchy (see Figure 4). The ideal end-effectors position and rotation were also different from the calculated one using the refurbish matrices.

10.2. Biped Model

For our test character, we used a 16 link biped model, shown in Figure 5. The character has 36 degrees of freedom (DOF). Character rigs can produce extremely non-linear motions due to their joint limits, flexibility and elaborate arrangement of joints.

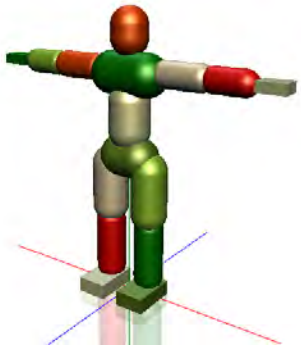


Figure 5. 16 link biped model used for testing.

Figure 5, shows the biped model in its starting stance pose.

Buildup of computational inaccuracies will cause a dual-quaternion to become of non-unit length; we fix these errors by renormalization. In contrast, repairing a non-orthogonal matrix is much more complicated (see [SALA79]).

11. RESULTS

The dual-quaternion unifies the translation and rotation into a single state variable. This single state variable offers a robust, unambiguous, computationally efficient way of representing rigid transform.

The computational cost of combining matrices and dual-quaternions:

Matrix4x4	: 64mult + 48adds
Matrix4x3	: 48mult + 32adds
DualQuaternion	: 42mult + 38adds

In our tests, we found the dual-quaternion multiplication method of transforms on average ten percent faster compared matrix multiplication. We did not take advantage of CPU architecture using parallel methods such as SIMD which can further improve speeds as demonstrated by Pallavi [MEHU10] (both for matrices and quaternion multiplication).

One major advantage we found when working with dual-quaternions was the added advantage of calculating angular and linear differences. When working with pure matrix methods we needed to convert the matrix to a quaternion to calculate angular variations.

12. CONCLUSION AND FURTHER WORK

The dual-quaternion model is an accurate, computationally efficient, robust, and flexible method of representing rigid transforms and should not be overlooked. Implementing pre-programmed dual-quaternion modules (e.g., multiplication and normalization) enables the creation of more elegant and clearer computer programs that are easier to work with and control.

While matrices are the *de-facto* method used for the majority of hierarchy based simulations, we have shown that they can present certain problems which are costly to avoid (e.g., renormalizing a matrix). The problem and cost of drifting and normalizing is less with dual-quaternions compared to matrix methods. When dealing with rigid transforms the dual-quaternion method shines through due to its numerous advantages.

This paper has only provided a taste of the potential advantages of dual-quaternions, and one can only imagine the further future possibilities that they can offer. For example, there is a deeper investigation of the mathematical properties of dual-quaternions (e.g., zero divisions). There is also the concept of dual-dual-quaternions (i.e., dual numbers within dual numbers) and calculus for multi-parametric objects for the reader to pursue if he desires.

13. APPENDIX

13.1. Dual-Quaternion Implementation Class.

```
public class DualQuaternion_c
{
public Quaternion   m_real;
public Quaternion   m_dual;
public DualQuaternion_c()
{
  m_real = new Quaternion(0,0,0,1);
  m_dual = new Quaternion(0,0,0,0);
}
public DualQuaternion_c( Quaternion r, Quaternion d )
{
  m_real = Quaternion.Normalize( r );
  m_dual = d;
}
public DualQuaternion_c( Quaternion r, Vector3 t )
{
  m_real   = Quaternion.Normalize( r );
  m_dual  = ( new Quaternion( t, 0 ) * m_real ) * 0.5f;
}
public    static    float    Dot(    DualQuaternion_c    a,
DualQuaternion_c b )
{
  return Quaternion.Dot( a.m_real, b.m_real );
}
public static DualQuaternion_c operator* (DualQuaternion_c
q, float scale)
{
  DualQuaternion_c ret = q;
  ret.m_real *= scale;
  ret.m_dual *= scale;
  return ret;
}
public static DualQuaternion_c Normalize( DualQuaternion_c q
)
{
  float mag = Quaternion.Dot( q.m_real, q.m_real );
  Debug_c.Assert( mag > 0.000001f );
  DualQuaternion_c ret = q;
  ret.m_real *= 1.0f / mag;
  ret.m_dual *= 1.0f / mag;
  return ret;
}
public static DualQuaternion_c operator + (DualQuaternion_c
lhs, DualQuaternion_c rhs)
{
  return new DualQuaternion_c(lhs.m_real + rhs.m_real,
    lhs.m_dual + rhs.m_dual);
}
// Multiplication order - left to right
public static DualQuaternion_c operator * (DualQuaternion_c
lhs, DualQuaternion_c rhs)
{
  return new DualQuaternion_c(rhs.m_real*lhs.m_real,
     rhs.m_dual*lhs.m_real + rhs.m_real*lhs.m_dual);
}
public static DualQuaternion_c Conjugate( DualQuaternion_c q
)
{
  return    new    DualQuaternion_c(    Quaternion.Conjugate(
q.m_real ), Quaternion.Conjugate( q.m_dual ) );
}
public static Quaternion GetRotation( DualQuaternion_c q )
{
  return q.m_real;
}
public static Vector3 GetTranslation( DualQuaternion_c q )
{
  Quaternion t = ( q.m_dual * 2.0f ) * Quaternion.Conjugate(
q.m_real );
  return new Vector3( t.X, t.Y, t.Z );
}
public      static      Matrix      DualQuaternionToMatrix(
DualQuaternion_c q )
{
  q = DualQuaternion_c.Normalize( q );

  Matrix M = Matrix.Identity;
  float w = q.m_real.W;
  float x = q.m_real.X;
  float y = q.m_real.Y;
  float z = q.m_real.Z;

  // Extract rotational information
  M.M11 = w*w + x*x - y*y - z*z;
  M.M12 = 2*x*y + 2*w*z;
  M.M13 = 2*x*z - 2*w*y;

  M.M21 = 2*x*y - 2*w*z;
  M.M22 = w*w + y*y - x*x - z*z;
  M.M23 = 2*y*z + 2*w*x;

  M.M31 = 2*x*z + 2*w*y;
  M.M32 = 2*y*z - 2*w*x;
  M.M33 = w*w + z*z - x*x - y*y;

  // Extract translation information
  Quaternion t = (q.m_dual * 2.0f) * Quaternion.Conjugate(
q.m_real);
  M.M41 = t.X;
  M.M42 = t.Y;
  M.M43 = t.Z;
  return M;
```

```
}

#if false
public static void SimpleTest()
{
  DualQuaternion_c     dq0     =     new     DualQuaternion_c(
Quaternion.CreateFromYawPitchRoll(1,2,3),                new
Vector3(10,30,90) );
  DualQuaternion_c     dq1     =     new     DualQuaternion_c(
Quaternion.CreateFromYawPitchRoll(-1,3,2),               new
Vector3(30,40, 190 ) );
  DualQuaternion_c     dq2     =     new     DualQuaternion_c(
Quaternion.CreateFromYawPitchRoll(2,3,1.5f),             new
Vector3(5,20, 66 ) );
  DualQuaternion_c dq = dq0 * dq1 * dq2;


  Matrix                    dqToMatrix                    =
DualQuaternion_c.DualQuaternionToMatrix( dq );

  Matrix   m0   =   Matrix.CreateFromYawPitchRoll(1,2,3)   *
Matrix.CreateTranslation( 10, 30, 90 );
  Matrix   m1   =   Matrix.CreateFromYawPitchRoll(-1,3,2)   *
Matrix.CreateTranslation( 30, 40, 190 );
  Matrix   m2   =   Matrix.CreateFromYawPitchRoll(2,3,1.5f)   *
Matrix.CreateTranslation( 5, 20, 66 );
  Matrix m = m0 * m1 * m2;
}
#endif
} // End DualQuaternion_c
```

13.2. Novice Errors

There are a few things to look out for when implementing a dual-quaternion class. Firstly, ensure the multiplication order is correct and remains consistent with matrices (i.e., left to right). Secondly, always ensure that the dual-quaternions remain normalized (i.e., unit-length).

14. REFERENCES

[CLIF82] W. Clifford, Mathematical Papers. London: Macmillan, 1882.

[KCŽO08] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, "Geometric skinning with approximate dual quaternion blending," ACM Transactions on Graphics (TOG), vol. 27, no. 4, p. 105, 2008.

[IVIV11] F. Z. Ivo and H. Ivo, "Spherical skinning with dual quaternions and QTangents," ACM SIGGRAPH 2011 Talks, vol. 27, p. 4503, 2011.

[SELI11] J. Selig, "Rational interpolation of rigid-body motions," Advances in the Theory of Control, Signals and Systems with Physical Modeling, pp. 213–224, 2011.

[VAFU09] A. Vasilakis and I. Fudos, "Skeleton-based rigid skinning for character animation," in Proc. of the Fourth International Conference on Computer Graphics Theory and Applications, 2009, no. February, pp. 302–308.

[KMLX11] Y. Kuang, A. Mao, G. Li, and Y. Xiong, "A strategy of real-time animation of clothed body movement," in Multimedia Technology (ICMT), 2011 International Conference on, 2011, pp. 4793–4797.

[PPAF10] H. L. Pham, V. Perdereau, B. V. Adorno, and P. Fraisse, "Position and orientation control of robot manipulators using dual quaternion feedback," in Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, 2010, pp. 658–663.

[SCHI11] M. Schilling, "Universally manipulable body models — dual quaternion representations in layered and dynamic MMCs," Autonomous Robots, 2011.

[GVMC98] Q. Ge, A. Varshney, J. P. Menon, and C. F. Chang, "Double quaternions for motion interpolation," in Proceedings of the ASME Design Engineering Technical Conference, 1998.

[LIWC10] Y. Lin, H. Wang, and Y. Chiang, "Estimation of relative orientation using dual quaternion," System Science and, no. 2, pp. 413-416, 2010.

[PEMC04] A. Perez and J. M. McCarthy, "Dual quaternion synthesis of constrained robotic systems," Journal of Mechanical Design, vol. 126, p. 425, 2004.

[ALMA92] W. Alan and W. Mart, Advanced Animation and Rendering Techniques: Theory and Practice. Adison-Wesley, 1992.

[GILB86] S. Gilbert, Introduction to Applied Mathematics. Wellesley-Cambridge Press, 1986.

[BERG09] G. van den Bergen, "Dual Numbers: Simple Math, Easy C++ Coding, and Lots of Tricks," GDC Europe, 2009. [Online]. Available: www.gdcvault.com/play/10103/Dual-Numbers-Simple-Math-Easy.

[HAMI86] W. R. Hamilton, Elements of Quaternions. London: , 1886.

[MCDO10] J. McDonald, "Teaching Quaternions is not Complex," Computer Graphics Forum, vol. 29, no. 8, pp. 2447-2455, Dec. 2010.

[SHOE85] K. Shoemake, "Animating rotation with quaternion curves," ACM SIGGRAPH computer graphics, 1985.

[SALA79] E. Salamin, "Application of quaternions to computation with rotations," Internal Report, Stanford University, Stanford, CA, vol. 1, 1979.

[MEHU10] P. Mehrotra and R. Hubbard, "Benefits of Intel® Advanced Vector Extensions For Quaternion Spherical Linear Interpolation (Slerp)," 2010. [Online]. Available: http://software.intel.com/en-us/articles/benefits-of-intel-advanced-vector-extensions-for-quaternion-spherical-liner-interpolation-slerp/.

An Efficient Preconditioner and a Modified RANSAC for Fast and Robust Feature Matching

Anders Hast
Uppsala University,
Uppsala, Sweden
anders.hast@it.uu.se

Andrea Marchetti
IIT, CNR
Pisa, Italy
andrea.marchetti@iit.cnr.it

ABSTRACT

Standard RANSAC does not perform very well for contaminated sets, when there is a majority of outliers. We present a method that overcomes this problem by transforming the problem into a 2D position vector space, where an ordinary cluster algorithm can be used to find a set of putative inliers. This set can then easily be handled by a modified version of RANSAC that draws samples from this set only and scores using the entire set. This approach works well for moderate differences in scale and rotation. For contaminated sets the increase in performance is in several orders of magnitude. We present results from testing the algorithm using the Direct Linear Transformation on aerial images and photographs used for panographs.

Keywords

RANSAC, Preconditioner, Homography, Clustering, Feature Matching, Image Stitching.

1 INTRODUCTION

RANSAC was introduced by Fischler and Bolles more than 30 years ago [FB81] and is one of the far most used algorithms for finding corresponding pairs of feature points in images. Distinguishing these so called true matches or inliers from the outliers or non matching pairs is essential for many applications of computer vision, such as image stitching [Sze10], 3D reconstruction [Pol00] and point-cloud shape detection [SWK07], just to mention a few. Many variants have been proposed since then, trying to enhance performance of the algorithm in different ways, as will be shown in the end of this section.

One disadvantage with standard RANSAC is that it handles contaminated sets poorly. In fact, many implementations of RANSAC do not perform well when the number of inliers is less than 50% [Low04]. RANSAC is based on random sampling, as the name itself suggests: RANdom Sample Consensus and the probability of finding an initial sample containing inliers only, decreases when the amount of outliers increases. Furthermore, RANSAC usually terminates when the probability of finding more inliers is low or rather when an outlier free set has been picked with some predefined probability. Nonetheless, for heavily contaminated sets, the output is not useful as it usually contains too few inliers if any. Moreover, the output set of putative inliers is often contaminated with outliers. Another consequence for highly contaminated sets is that the stopping criterion might indicate that there is not yet a consensus, while most of the inliers are already found.

Contributions and Delimitations

We propose a naive preconditioner that eliminates the majority of outliers before running a modified version of LO-RANSAC [CMK03] on the set. The preconditioner transforms the problem of finding the consensus set to a position vector space, where an ordinary clustering algorithm can be used to find the cluster that contains the putative inliers. It will be show in examples that the approach works well if the differences in rotation and scale are moderate, which they usually are for matching of images with mainly sideway camera translations. The modified RANSAC will draw samples from this set only and whenever a larger set is found the local optimization step samples this set at least 4 times while using the homography to score the whole set. This approach will be many times faster for contaminated sets than ordinary RANSAC, as the transformation is simple and clustering is relatively fast. Moreover, the modified RANSAC will find consensus in very few iterations as it works on a set with a large majority of inliers. The preconditioner will therefore reduce the number of iterations in the modified RANSAC by orders of magnitude for contaminated sets. Since clustering can be done with $O(n)$ complexity it could also be used for sets with low contamination as it will be fast. However, In this paper an $O(n^2)$ algorithm was used.

The proposed approach will be compared to standard RANSAC only, as many of the already proposed extensions of RANSAC could be used to enhance

the modified RANSAC. Especially, MultiRANSAC [ZKM05] could be used when there are multiple planes in the images. Nonetheless, the proposed approach is able to handle such cases too and it will be discussed how.

Furthermore, we have chosen to delimit ourselves in this paper to use a perspective transformation based on at least four points, the so called Direct Linear Transformation (DLT)[HZ03], which throughout the text we will be referred to as the homography. This transformation can be used for a number of applications such as image stitching of aerial images and panographs.

RANSAC and some of its Variants

Standard RANSAC proceeds in the following way: first a minimal number of points is selected, which is required to determine the homography [BL07] [HZ03] [VL01], which is the projective transformation between the images. Then the set is scored so that the inliers that falls below a certain predefined tolerance ε are counted. After transforming using the homography, these points are close enough to its corresponding match and are therefore regarded as true inliers. The algorithm terminates when the probability of finding a better model falls under a predefined threshold, otherwise it starts all over.

RANSAC generally treats all correspondences equally and draws random samples uniformly from the full set. However there are some approaches that tries to exclude probable outliers early on or alternatively determining which ones are probable inliers. Just to mention a few: MLESAC [TZ00] performs non-uniform, i.e. guided sampling of correspondences and PROSAC [CM05] draw samples from a progressively larger set of top-ranked correspondences. GODSAC [MvHK*06] use an assessment driven selection of good samples, instead of random sampling. Fuzzy RANSAC [LK07] divides the input data into categories depending on the residual error and sampling is done in the good set. Another approach [ZK06] transforms the whole problem into classification of the residual distribution. SCRAMSAC [SLK09] tries to reduce the number of outliers using a spatial consistency check. R-RANSAC [CM08], was proposed for the situation when the contamination of outliers is known, using a randomized model verification strategy. Cov-RANSAC [RFP09] incorporates the inherent uncertainty of the estimation procedure in order to achieve a more efficient algorithm. GroupSAC[NJD09] take advantage of additional grouping information between features provided by optical flow based clustering.

Other approaches, designed for real-time tracking take into account that there are similarities between a series of images captured by a camera. Hence, the order of the scoring of the pairs of matches can be planned in order to avoid scoring useless pairs. KALMANSAC [VJFS05] was designed for estimation of structure from motion (SFM). It is derived from pseudo-Bayesian filtering algorithms in a sampling framework and can handle sequences containing large number of outliers. Other examples from robotics are Preemptive RANSAC [Nis03] and Iterative RANSAC [KK06].

Other important contributions to RANSAC use different strategies. MultiRANSAC [ZKM05] is a parallel version that allows to deal with multiple models and have the advantage of being able to cope with a high percentage of outliers. GASAC [RH06] is another parallel approach using a genetic algorithm approach. Moreover, RANSAC has a low probability to find the correct solution when the data is quasi degenerate and QDEGSAC [FP05] was proposed for use in such cases. NAPSAC[MTN*02] was proposed for problems with high noise and takes advantage of the fact that if an inlier is found then any point close to that point will have a high probability to be an inlier.

There are many more versions proposed in literature and a performance evaluation of some of the more important variants of RANSAC is done by Choi et al. [CKY09] and a comparative analysis of RANSAC is given by Raguram et al. [RFP08]. Lowe [Low04] proposed to use the Hough transform [DH72] for clustering data instead of RANSAC, and there are even hybrids [HH07]. Nevertheless, RANSAC is after more than 30 years still used and improved for computer vision applications.

2 THE PRECONDITIONER

The idea is to use a preconditioner that transforms the problem of finding the consensus set to finding a cluster in a position vector space. Generally, a vector can be constructed from two points and each matching pair consists of exactly two points. Hence, it can be regarded as a vector from image a to image b, just like how the final homography transforms each point in image a to its corresponding point in image b, within a certain threshold ε. The main advantage is that a position vector can be treated as a 2D point rather than a 2D vector. The position vector will be scaled in the range $[0..1]$ so that the cluster algorithm can be given a tolerance ε_c similar to the tolerance ε used for the modified RANSAC. This is done by dividing the vector by the length of the sum of the sides in each direction, where image b is translated in each direction using the lengths of image a, so that there is no spatial overlap between the images.

Let the position vector between the feature point at (x_0, y_0) in image a and the corresponding point (x_1, y_1) in image b be:

$$\mathbf{v} = \left[\frac{(a_x + x_1) - x_0}{a_x + b_x}, \frac{(a_y + y_1) - y_0}{a_y + b_y} \right], \qquad (1)$$

where a_x, a_y and b_x, b_y are the sizes in the x and y direction for image a and b, respectively.

The nice result of such an approach is that true matches will yield points in the 2D space that are forming a cluster, while outliers will be spread out in a more random fashion. The search for true matches can therefore be done using any appropriate 2D clustering algorithm, since the vectors are regarded as points rather than vectors, i.e. they are position vectors. This is true also for cases when the images are taken from a sequence of a forward camera motion. Some prefer to visualize the matching by showing only one of the images using lines that start in the feature points in that image. The line ends in the point corresponding to the feature points of the second image, which is not shown but is supposed to overlap the first image. If the images are taken from a sequence of a forward camera motion this approach will yield lines that can point in independent directions. However, if the proposed approach is used, where the images are put so that they do not overlap and they do not share an edge, then they will all have a similar direction compared to the outliers. The direction and length of these lines or vectors, will not be exactly the same and can vary. Nonetheless, they will usually vary a lot less than compared to the outliers even if the cluster will be less dense.

Clustering

There are many clustering algorithms [CRW91] [JMF99] that could be used and some of the more popular are k-means clustering [Mac67] and the mean shift algorithm [CM02]. In our tests it was chosen to use a simple approach that for each point (position vector) in the set computes the distance to all other points. The point that have most neighbors closer than the threshold ε_c will be chosen as the cluster center and all points in the cluster are considered putative inliers. Obviously a better algorithm could be used, especially for situations where the points lie in different planes giving different homographies. However, focus in this paper does not lie on the clustering algorithm as it is a well studied area. Hence, we will instead focus on the preconditioner that transforms the matches to the new position vector space and on how to treat the clustered points using a modified version of LO-RANSAC.

The cluster will contain a majority of the inliers and also some outliers depending both on the tolerance ε_c and how well the cluster algorithm performs. Nonetheless, it is not of vital importance that the cluster will contain inliers only, as the modified RANSAC will clean it up. In all our tests we used the same value of ε_c for the clustering as the tolerance ε for the modified RANSAC.

The computational cost for the preconditioner is rather low. We used a simple approach to find the cluster center. First the vectors are computed and then the clustering algorithm needs to find the cluster. The cost of computing the distance between all points in the space for a brute force algorithm is $n(n-1)/2$, hence the complexity is $O(n^2)$. Then all points sufficiently near the point with most neighbors need to be found. Nevertheless, this cost could be reduced by dividing the space using for instance quad trees [FB74] or kd-trees [TBK08]. Moreover, binning would reduce the complexity to $O(n)$ as each point is classified to belong to a bin depending on its spatial location, in a linear search. The bin with most points will be chosen as the cluster. Nonetheless, the borders of the bins may divide the cluster and this can easily be handled by overlapping bins. Once again the bin with most points are the putative inliers. The size of the bins would be proportional to the tolerance ε_c.

A Modified RANSAC

A modified version of the LO-RANSAC [CMK03] [CMO04] algorithm is here proposed, which utilizes a local optimization step. Both the cluster and the whole set are input parameters to the algorithm, which samples from the cluster only. As the cluster contains the set, which is close to the final solution and therefore pretty free from outliers, it was chosen to sample using up to half of the matches in the cluster but obviously never less than four. This usually lead to consensus faster than sampling just four samples every time, which give less support compared to using up to half of them. This set is used to estimate the homography and scoring the number of inliers.

Every time scoring gives a maximum set of inliers the local optimization step samples iteratively from this set and estimates the homography from it. However, scoring is done using the whole set. Once again it is more efficient to use up to half the size of the set, when doing re-estimation and re-scoring. Whenever a larger set is found it uses this set to sample from and restarts the local optimization loop. We have found that about 4 iterations is usually enough for the proposed approach, while Chum et al used 10 iterations. This is of course a value that can be increased if necessary. The algorithm terminates when the probability is 99% that we have picked an outlier free set and the parameters for this test are constructed using the set belonging to the cluster. Generally, N iterations are need in order to find an outlier free set with the probability p as:

$$N = \frac{\log(1-p)}{\log(1-\gamma^s)}, \tag{2}$$

where γ is the inlier ratio, i.e. number of inliers divided by number of points in the cluster and s is the number of samples drawn each time.

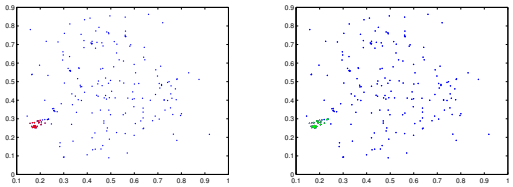


Figure 1: The result of using the preconditioner of a quasi degenerate set, where all points in blue are outliers. Left: the cluster found by the clustering algorithm (red). Right: the points in the space that corresponds to the true inliers found by the modified RANSAC (green).

If N is larger than the number of iterations of the main loop the algorithm starts all over and samples the cluster set from the preconditioner once again.

3 RESULTS

Several tests using different images were conducted in order to prove the efficiency of the proposed preconditioner and the modified RANSAC. The Harris corner detector [HS88] was chosen to detect features in most of the tests instead of the more accurate SIFT detector [Low04]. As the proposed method will be of interest especially for sets with rather high contamination, at least 50%, Harris is preferable as it is less accurate than SIFT.

A Quasi Degenerate Set

A quasi degenerate set with an inlier ratio of 0.1651 was chosen and according to equation 2 it would need 6196 iterations to find an outlier free set. The perspective distortion is small in these aerial images. However, they are rotated in a way that it becomes very hard for standard RANSAC to find the consensus set. A test was performed 10 000 times measuring how many iterations were needed to find *all* inliers and the result was on average 32 887 iterations. Moreover, a test was done 10 000 times counting both number of iterations and number of inliers using the preconditioner and the modified RANSAC. The result is shown in Table 1 on the first row. The preconditioner finds the set (red) in Fig. 1 at the left. In the right is the same points with the inliers (green) that are found by the modified RANSAC.

The images used for the result in Fig. 1 are shown in Fig. 2. The true inliers are connected by yellow lines and the outliers with red ones. The set is quasi degenerate because the true matches covers a small area, which is rather elongated. Furthermore, the images are not perfectly aligned with each other as there is about 18 degrees of rotation between them. This causes standard RANSAC to find just a portion of the inliers in most runs. This problem is overcome by the preconditioner as it finds the major part of the inliers and the modified RANSAC draws sample from this set.

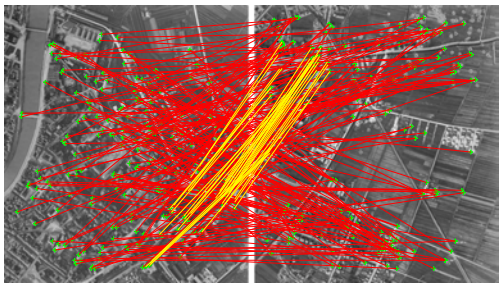


Figure 2: ©MiBAC-ICCD, Aerofototeca Nazionale, fondo RAF. Two historical photos taken over Pisa during WWII, with the true inliers connected with yellow lines (16.5%) and the false matches with red lines (83.5%)

	Iterations	Inliers			
	N	μ	σ	μ	σ
1	6196	6.144	0.452	36.000	0.000
2	1123000	17.904	6.967	81.005	0.261
3	87	6.000	6.000	211.000	0.000
4	8271	11.103	5.747	46.999	0.150
5	26	7.000	0.000	225.000	0.000
6	731	8.817	2.878	76.723	0.828
7	9	6.000	0.000	137.000	0.000
8	90	7.934	3.573	11.954	2.646
9	18	6.005	0.071	90.000	0.000

Table 1: The number of iterations (theoretical) and the mean and standard deviation for number of iterations and inliers for different matchings and images.

A Heavily Contaminated Set

A heavily contaminated set with just an inlier ratio of 0.045 was obtained by increasing the number of feature points and the ratio for the matching. Figure 3 shows how the preconditioner finds the cluster (red) in the image at the left. In the right is a close-up of the inliers (green). Standard RANSAC would, according to equation 2, need about 1 123 000 iterations to find the majority of inliers. After the preconditioner, the probability is increased to 0.9759, which corresponds to 1.93 iterations on average. The modified RANSAC could easily find almost all inliers in every run in just about 17 iterations as shown on the second row in Table 1 which is an enormous increase in performance compared to the theoretical 1 123 000 iterations.

An Almost All Inlier Set

A set that is almost outlier free with an inlier ratio is 98.40% was tested and Figure 4 shows how the preconditioner finds the whole set (red). The theoretical number of iterations are just 1.66 and the modified RANSAC needs 6 iterations to find the set, which

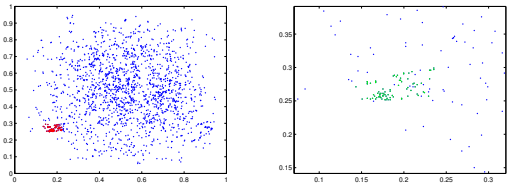


Figure 3: The result of using the preconditioner of a highly contaminated set, where all points in blue are outliers. Left: the cluster found by the clustering algorithm (red). Right: a close up of the points in the space that corresponds to the true inliers found by the modified RANSAC (green).

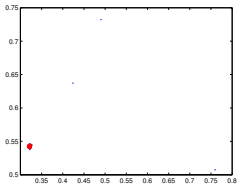


Figure 4: A set with an inlier ratio of 98.40% **The preconditioner finds the cluster (red), which is the same set as the modified RANSAC will find.**

could be reduced by diminishing the number of iterations in the local optimization.

Multiple Planes

A set of photos taken on ground were used to test the algorithm for stitching of panographs. The preconditioner is also able to find the correct cluster for images where the perspective distortion is greater and as in this case, where there are several planes. The preconditioner and modified RANSAC was used for the set of images shown in Figure 5 and Figure 6 shows how the preconditioner finds the cluster (red) in the image to the left and on the right is a close-up of the inliers (green). The cluster becomes elongated and curved because of perspective distortions and the three planes in the image. Nevertheless, the clustering algorithm is able to find the cluster containing all three planes. A more sophisticated clustering algorithm might be able to separate it into three clusters. However, this task could also be handled by a version of RANSAC that finds multiple planes.

The modified RANSAC finds all inliers in every run in just 6 iterations as shown on the third row in Table 1 compared to the theoretical value of 87 iterations for finding 99% of the inliers. This is not a huge increase in performance. However, keep in mind that the modified RANSAC finds all inliers in every run for this set, which is not the case for standard RANSAC. Increasing the rate to 99.99% would double the theoretical number of iterations needed.

Another set of images were used in the next test and Figure 7 shows how the preconditioner finds the cluster (red) at the left. In the right is a close-up of the

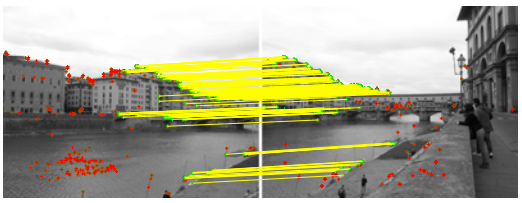


Figure 5: ©Anders Hast. Two images of the "Ponte Vecchio" in Florence, Italy. The inliers are connected with yellow lines (47.63%) and the false matches are depicted with red crosses (52.37%)

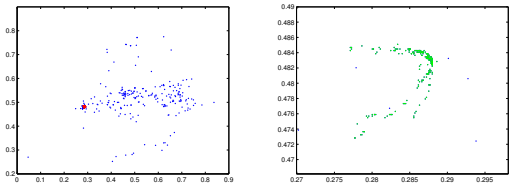


Figure 6: A moderately contaminated set with an inlier ratio of 47.63% **Left: The preconditioner finds the cluster (red). Right: A close up of the inliers found by the modified RANSAC (green).**

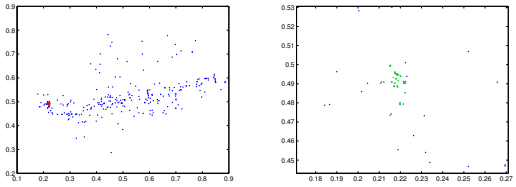


Figure 7: A contaminated set with an inlier ratio of 15.36% **Left: The preconditioner finds the cluster (red). Right: A close up of the inliers found by the modified RANSAC (green).**

inliers (green). The modified RANSAC finds almost all inliers in every run in just 11 iterations as shown on the fourth row in Table 1 compared to the theoretical value of 8271 iterations.

Yet another set of images where used and Figure 8 shows how the preconditioner finds the cluster (red) at the left. In the right is a close-up of the inliers (green). The modified RANSAC finds almost all inliers in every run in just 7 iterations as shown on the fifth row in Table 1 compared to the theoretical value of 26 iterations. Standard RANSAC found all inliers in an average of 157.6 iterations with a standard deviation of 156.4. Hence, the proposed method have the advantage of being less variable when it comes to the number of iterations, also for medium contaminated sets.

Finally we made a test using SIFT on a pair of images where there are two separate planes as shown in Figure 9. This time SIFT was used since Harris was not able to detect the points we were interested in. One set of inliers is connected with yellow lines and the other with blue ones. The cluster found for the yellow

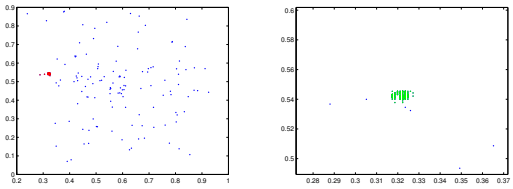


Figure 8: A set with an inlier ratio of 63.38% Left: The preconditioner finds the cluster (red). Right: A close up of the inliers found by the modified RANSAC (green).

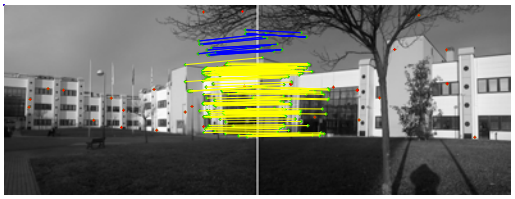


Figure 9: ©Anders Hast Two images with two clearly separable planes. The major set of inliers are connected with yellow lines and the minor set of inliers with blue ones.

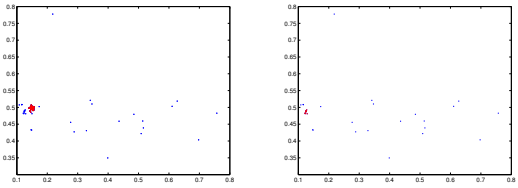


Figure 10: The preconditioner finds the first plane corresponding to the cluster (left). RANSAC finds at the inliers corresponding to that cluster and it is removed from the set. The preconditioner finds the second plane (right) corresponding to the cluster (red)

ones is shown in the left of Figure 10 and the other cluster in the right. Since the clusters were easily separated the preconditioner and modified RANSAC was run to find the first set of inliers. Then these were removed from the whole set and the procedure was repeated. The next set of inliers was easily found by the proposed approach. The result is on row six and seven in the Table 1 for each set.

Scale Differences

One aerial image was scaled down to 75% of its size to examine the impact on the cluster. As can be seen from Figure 11 the shape of the cluster is different from the one in Figure 8 even if they are exactly the same images. Depending on the cluster approach used and the tolerance ε_c, the preconditioner will find such sets as well and the modified RANSAC has no problems of handling them. (See row eight in Table 1.)

In the second test a pair of images, shown in Figure 12 were matched using SIFT in order to obtain

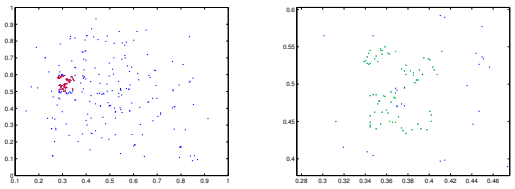


Figure 11: The Cluster (left) becomes larger in size when the scale in the input images are different. To the right is the inliers (green).

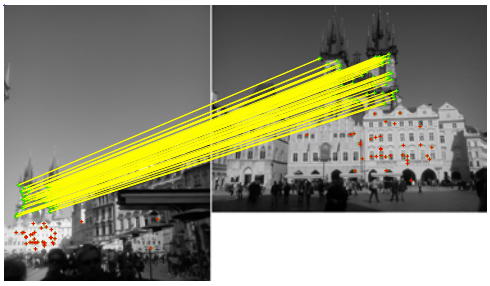


Figure 12: ©Anders Hast. Two images with taken on different distances from the main object (the chuch towers). The set of inliers are connected with yellow lines and the outliers are depicted with red crosses.

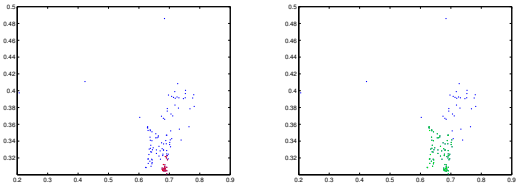


Figure 13: A set with an inlier ratio of 63.38% Left: The preconditioner finds the cluster (red). Right: A close up of the inliers found by the modified RANSAC (green).

better matches of the images that were taken on different distances to the object. Hence, the same problem of scale will occur. Figure 13 shows how the preconditioner finds just a part of the set (depending on the tolerance ε_c), which once again becomes more spread over a larger area. Anyhow, the modified RANSAC will find the whole set and the result is on row nine in Table 1.

Efficiency

Some further testing were done to test the efficiency of the method and the results are shown in Table 2. Four sets of aerial images (four first rows) and six sets of photos taken on the ground (rows five to ten) were used. The tests were once again performed 10 000 times. In the first column is the size of the clusters obtained by the preconditioner. Next is the theoretical number of iterations (N). Then follows the mean μ and standard deviation σ of the number of iterations needed by the modified RANSAC to find the inliers.

	Cluster Size	Iter. N	μ	σ	Inl. μ	σ
1	559	19	7.5	1.5	551.0	0.09
2	82	569	28.9	11.9	73.6	4.9
3	37	9144	7.9	2.3	36.7	0.48
4	50	$2.3 \cdot 10^4$	14.5	6.2	49.0	0.7
5	221	61	10.9	1.9	196.9	0.5
6	141	214	19.6	5.1	119.8	0.97
7	105	891	25.7	10.5	101.0	0.25
8	61	6109	20.6	7.7	53.1	1.3
9	44	$6.0 \cdot 10^4$	6.0	0.03	56.0	0.0
10	56	$2.4 \cdot 10^6$	6.2	0.5	67.0	0.09

Table 2: Four test runs for aerial images and six for images with multiple planes, with the cluster size, the mean and standard deviation for number of iterations and inliers.

The next values in the end of the row is the μ and σ of number of inliers.

The proposed approach is able to find most of the inliers with low deviation, except for the case on row 2 and 8, which have a σ greater than 1.0. Remember that only 4 iterations are done in the local optimization step and the σ could be decreased by increasing this number, which of course would increase the number of iterations in total.

Obviously, the proposed approach is very efficient as it reduces the number of iterations while still maintaining a high accuracy in terms of number of inliers found. Most remarkably is that the preconditioner makes it, not only possible, but even easy and fast to find the consensus set when the theoretical number of iterations exceeds tenth's of thousands and even millions. The result on row ten is from a set with an inlier ratio of 0.0372 and the theoretical number of iterations exceeds 2.4 million iterations. By using the preconditioner the number of iterations were 6.2 on average with only 0.5 in deviation. All 67 inliers were found in almost every run with a deviation of only 0.09. When the proposed approach does not find all inliers in every run one could increase the number of iterations in the local optimization step to increase the probability of finding more inliers.

4 DISCUSSION

It is important to set an appropriate tolerance ε for RANSAC and likewise it is important that the tolerance ε_c is set properly for the preconditioner. By scaling the position vector into the range $[1..0]$ it is possible to use the same tolerance for both. Nonetheless, care must be taken so that the tolerance is proportional to the size of the image. Moreover, one must take into account the scale differences as it will affect the size of the cluster and the tolerance must be set accordingly. A similar case is when the images are taken during a forward camera movement, which yields images with different scales. It has been shown that the preconditioner is able to handle moderate changes in scale, even if only a part of the cluster is found because the cluster becomes proportionally larger, i.e it is spread out.

When there are multiple planes in the image, the cluster will be a bit different and sometimes it is even separable in space, but not always. Here some more sophisticated clustering algorithm could be used in order to separate the clusters in a more accurate way. Nonetheless, the preconditioner was able to find the main cluster in all our tests and the modified RANSAC extracted all inliers from the set. Hence, it is possible to modify and use some other version of RANSAC that is able to yield separate planes such as Multi-RANSAC [ZKM05]. Otherwise, one could also in many cases extract one cluster at a time and run the modified RANSAC on each of them.

The size of the cluster will also affect the result and different ε_c could be tested. Moreover, it is possible to change the performance by changing how many samples are drawn. Usually four samples are drawn in standard RANSAC. However, by increasing this number to half of the current consensus set, but obviously never less than four, performance was increased for the modified RANSAC. One could experiment further with what is actually the optimal number to use.

We delimited ourselves to use the four point DLT. Nonetheless, there is nothing that prevent using other types of homographies. In any case, the output of the preconditioner is independent of the homography. It is just the result of the modified RANSAC that might change depending on what homography is being used. Moreover, we used a clustering algorithm that was easy to implement but is not the fastest one. Nevertheless, what clustering algorithm to use is not so important. The important thing is that it finds the cluster and preferably does that fast.

5 CONCLUSION AND FUTURE WORK

Standard RANSAC handles highly contaminated sets poorly as the probability of drawing samples giving an outlier free set after scoring becomes very small. This problem can easily be overcome by the proposed preconditioner that transforms the problem to a position vector space where each vector is a scaled vector representing the matches. An ordinary clustering algorithm can be used to find the cluster of putative inliers. This set is then processed by a modified version of RANSAC that draws from this set exclusively but scores using the whole set. This approach will increase performance substantially for contaminated sets. The preconditioner can be used also for sets with low con-

tamination as the clustering algorithm is relatively fast compared to estimation and scoring in the RANSAC procedure.

The preconditioner can be modified in such way that more powerful clustering methods are used in order to find more than one projection plane. Moreover, it should be determined how large differences in scale and rotation the preconditioner can handle and also what could be done to handle the more extreme cases.

6 REFERENCES

[BL07] Brown M., Lowe D. G.: Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision 74*, 1 (2007), 59–73.

[CKY09] Choi S., Kim T., Yu W.: Performance evaluation of ransac family. In *British Machine Vision Conference* (2009), pp. 1–12.

[CM02] Comaniciu D., Meer P.: Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis Machine Intelligence (PAMI) 24*, 5 (2002), .603–619.

[CM05] Chum O., Matas J.: Matching with prosac - progressive sample consensus. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 220–226.

[CM08] Chum O., Matas J.: Optimal randomized ransac. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 8 (2008), 1472–1482.

[CMK03] Chum O., Matas J., Kittler J.: Locally optimized ransac. In *the Annual Pattern Recognition Symposium of the German Association for Pattern Recognition (DAGM)* (2003), pp. 236–243.

[CMO04] Chum O., Matas J., Obdrzalek S.: Enhancing ransac by generalized model optimization. In *Asian Conference on Computer Vision (ACCV)* (2004).

[CRW91] Capoyleas V., Rote G., Woeginger G.: Geometric clusterings. *Journal of Algorithms 12* (1991), 341–356.

[DH72] Duda R. O., Hart P. E.: Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM 15* (1972), 11–15.

[FB74] Finkel R. A., Bentley J. L.: Quad trees a data structure for retrieval on composite keys. *Acta Informatica 4*, 1 (1974), 1–9.

[FB81] Fischler M. A., Bolles R. C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24* (1981), 381–395.

[FP05] Frahm J. M., Pollefeys M.: Ransac for (quasi-) de-generate data (qdegsac). In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 220–226.

[HH07] Hollander R. J. M. D., Hanjalic A.: A combined ransac-hough transform algorithm for fundamental matrix estimation. In *British Machine Vision Conference* (2007).

[HS88] Harris C., Stephens M.: A combined corner and edge detection. In *Alvey Vision Conference* (1988), pp. 147–151.

[HZ03] Hartley R. I., Zisserman A.: *Multiple View Geometry â 2nd edition*. Cambridge University Press, 2003.

[JMF99] Jain A., Murty M., Flynn P.: Data clustering - a review. *ACM Computing Surveys 31*, 3 (1999), 264–323.

[KK06] K K. T., Kondo E.: Incremental ransac for online relocation in large dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)* (2006), pp. 1025–1030.

[LK07] Lee J. J., Kim G.: Robust estimation of camera homography using fuzzy ransac. In *Proceedings of the 2007 international conference on Computational science and its applications - Volume Part I* (Berlin, Heidelberg, 2007), ICCSA'07, Springer-Verlag, pp. 992–1002.

[Low04] Lowe D. G.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[Mac67] MacQueen J. B.: Some methods for classification and analysis of multivariate observations. In *5-th Berkeley Symposium on Mathematical Statistics and Probability* (1967), vol. 1, Berkeley, University of California Press, pp. 281–297.

[MTN*02] Myatt D., Torr P., Nasuto S., Bishop J., Craddock R.: Napsac: High noise, high dimensional robust estimation - its in the bag. In *British Machine Vision Conference* (2002), vol. 2, pp. 458–467.

[MvHK*06] Michaelsen E., von Hansen W., Kirchhof M., Meidow J., Stilla U.: Estimating the essential matrix: Goodsac versus ransac. In *Photogrammetric Computer Vision* (2006), pp. 1–6.

[Nis03] Nister D.: Preemptive ransac for live structure and motion estimation. In *International Conference on Computer Vision (ICCV)* (2003), pp. 109–206.

[NJD09] Ni K., Jin H., Dellaert F.: Groupsac: Efficient consensus in the presence of groupings. In *ICCV* (2009), IEEE, pp. 2193–2200.

[Pol00] Pollefeys M.: Automated reconstruction of 3d scenes from sequences of images. *ISPRS Journal of Photogrammetry and Remote Sensing 55*, 4 (2000), 251–267.

[RFP08] Raguram R., Frahm J.-M., Pollefeys M.: A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *European Conference on Computer Vision (ECCV)* (2008), pp. 500–513.

[RFP09] Raguram R., Frahm J.-M., Pollefeys M.: Exploiting uncertainty in random sample consensus. In *International Conference on Computer Vision (ICCV)* (2009), pp. 2074–2081.

[RH06] Rodehorst V., Hellwich O.: Genetic algorithm sample consensus (gasac) - a parallel strategy for robust parameter estimation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)* (2006), pp. 1–8.

[SLK09] Sattler T., Leibe B., Kobbelt L.: Scramsac: Improving ransac's efficiency with a spatial consistency filter. In *International Conference on Computer Vision (ICCV)* (2009), pp. 2090–2097.

[SWK07] Schnabel R., Wahl R., Klein R.: Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (2007), 214–226.

[Sze10] Szeliski R.: Computer vision : Algorithms and applications. *Computer 5*, 3 (2010), 832.

[TBK08] Tsakok J. A., Bishop W., Kennings A.: kd-tree traversal techniques. *2008 IEEE Symposium on Interactive Ray Tracing 44*, 1 (2008), 190–190.

[TZ00] Torr P. H. S., Zisserman A.: Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding 78* (2000), 138–156.

[VJFS05] Vedaldi A., Jin H., Favaro P., Soatto S.: Kalmansac: Robust filtering by consensus. In *International Conference on Computer Vision (ICCV)* (2005), pp. 633–640.

[VL01] Vincent E., Laganiere R.: Detecting planar homographies in an image pair. *Image and Signal Processing and Analysis* (2001), 182–187.

[ZK06] Zhang W., Kosecka J.: A new inlier identification scheme for robust estimation problems. In *Proceedings of Robotics: Science and Systems* (Philadelphia, USA, August 2006).

[ZKM05] Zuliani M., Kenney C., Manjunath B.: The multiransac algorithm and its application to detect planar homographies. In *The International Conference on Image Processing (ICIP)* (2005), vol. 3, pp. 153–156.

Fast GPU Garment Simulation and Collision Detection

Tzvetomir I. Vassilev
Dept. of IIT, University of Ruse
8 Studentska St
Bulgaria 7017, Ruse
tvassilev@uni-ruse.bg

Bernhard Spanlang
EventLab, Universitat de Barcelona
Campus de Mundet - Edifici Teatre
Passeig de la Vall d'Hebron 171,
Spain 08035, Barcelona
bspanlang@ub.edu

ABSTRACT

This paper describes a technique for garment simulation and collision detection implemented on modern Graphics Processors (GPU). It exploits a mass-spring cloth model with velocity modification approach to overcome the super-elasticity. Our novel algorithms for cloth-body and cloth-cloth collision detection and response are based on image-space interference tests. For collision detection a 3D texture is generated, in which each slice represents depth and normal information for collision detection and response. These algorithms were implemented to build a fast web-based virtual try-on system. Our simulation starts from flat garment pattern meshes and performs the entire seaming and cloth draping simulation on the GPU. By mapping cloth properties of real fabric measurements we are able to approximate the real drape behaviour of different types of fabric, taking into account different warp and weft fabric drape properties. As the results section shows the average time of dressing a virtual body with a garment on state of the art graphics hardware is 0.2 seconds.

Keywords

Cloth Simulation, GPU programming, Collision detection.

1. INTRODUCTION

Physical simulation and elastic deformable objects have been widely used by researchers in computer graphics. The main applications of garment simulation are in the entertainment industries, in the fashion design industry and in electronic commerce when customers shop for garments on the web and try them on in a virtual booth.

The graphics processing unit (GPU) on today's commodity video cards has evolved into an extremely powerful and flexible processor [LHG*06]. The latest graphics architectures provide huge memory bandwidth and computational power, with fully programmable vertex and pixel processing units that support vector operations up to full IEEE floating point precision. Architecturally, GPUs are highly parallel streaming processors optimized for vector operations, with single instruction on multiple data (SIMD) pipelines. Not surprisingly, these processors are capable of general-purpose computation beyond the graphics applications for which they were designed and many researchers have utilized them in cloth modelling [Zel05], [GW05].

2. BACKGROUND

Previous work in cloth simulation

Physically based cloth modelling has been a problem of interest to computer graphics researchers for more than two decades. First steps, initiated by Terzopoulos et al. [TPBF87], characterised cloth simulation as a problem of deformable surfaces and used the finite element method and energy minimisation techniques borrowed from mechanical engineering. Since then other groups have been formed [BHW94], [EWS96], [CYTT92] challenging the cloth simulation using energy or particle based methods.

Provot [Pro95] used a mass-spring model to describe rigid cloth behaviour, which proved to be faster than the techniques described above. Its major drawback is the super-elasticity. In order to overcome this problem he applied a position modification algorithm to the ends of the over-elongated springs. However, if this operation modifies the positions of many vertices, it may elongate other springs. Vassilev et al. [VSC01] used a velocity modification approach to solve the super-elasticity problem.

The nature of the mass-spring system is suitable for implementation on the GPU. NVIDIA [Zel05] have provided a free sample demo of a mass-spring cloth simulation on their graphics processors. Their cloth model is quite simple and does not simulate resistance to bending. Rodriguez-Navarro et al. have published two implementations of cloth model on the GPU. The first is based on a mass-spring system [RNSS05] and the second on the finite element method [RNS06]. Georgii and Westermann [GW05] compared two possible implementations of mass-spring systems on the GPU and tested them with cloth simulation. A GPU accelerated mass-spring system has been used in other fields like surgical simulation [MHS05]. The advantage of the mass-spring system, described in this paper, is that it implements on the GPU a velocity modification approach for overcoming the super-elasticity, which results in a faster and more realistic simulation. Unlike other GPU implementations we perform also the garment seaming process on the GPU and we use fabric property measurements in order to approximate the behaviour of real fabric to a good degree.

Mass-spring model of cloth

The method, described in this work is based on the cloth model described in [VSC01]. The elastic model of cloth is a mesh of $l \times n$ mass points, each of them linked to its neighbours by massless springs of natural length greater than zero. There are three different types of springs: structural, shear, and flexion, which implement resistance to stretching, shearing and bending, correspondingly.

Let $\mathbf{p}_{ij}(t)$, $\mathbf{v}_{ij}(t)$, $\mathbf{a}_{ij}(t)$, where i=1,…, l and j=1,…, n, be correspondingly the positions, velocities, and accelerations of the mass points at time t. The system is governed by Newton's basic law:

$$\mathbf{f}_{ij} = m\,\mathbf{a}_{ij}, \qquad (1)$$

where m is the mass of each point and $\mathbf{f}_{ij}$ is the sum of all forces applied at point $\mathbf{p}_{ij}$. The force $\mathbf{f}_{ij}$ can be divided in two categories.

The **internal forces** are due to the tensions of the springs. The overall internal force applied at the point $\mathbf{p}_{ij}$ is a result of the stiffness of all springs linking this point to its neighbours:

$$f_{int}(p_{ij}) = -\sum_{k,l} k_{ijkl}\left(\overrightarrow{p_{kl}p_{ij}} - l^0_{ijkl}\frac{\overrightarrow{p_{kl}p_{ij}}}{\left\|\overrightarrow{p_{kl}p_{ij}}\right\|}\right), \qquad (2)$$

where k_{ijkl} is the stiffness of the spring linking $\mathbf{p}_{ij}$ and $\mathbf{p}_{kl}$ and $l$is the natural length of the same spring.

The **external forces** can differ in nature depending on what type of simulation we wish to model. The most frequent ones are gravity and viscous damping.

All the above formulations make it possible to compute the force $\mathbf{f}_{ij}(t)$ applied on cloth vertex $\mathbf{p}_{ij}$ at any time t. The fundamental equations of Newtonian dynamics can be integrated over time by a simple Euler, Verlet or Runge-Kutta method [PTVF92].

Collision detection

Collision detection (CD) and response prove to be the bottleneck of dynamic simulation algorithms that use highly discretised surfaces. Most CD algorithms between cloth and other objects in the scene are based on geometrical object-space (OS) interference tests. Some apply a prohibitive energy field around the colliding objects [TPBF87], but most of them use geometric calculations to detect penetration between a cloth particle and a triangle of the object together with techniques that reduce the number of tests.

Most common approaches are voxel or octree subdivision [Gla98]. Another solution is to use a bounding box (BB) hierarchy [BW98], [Pro97]. Objects are grouped hierarchically according to proximity rules and a BB is pre-computed for each object. The collision detection is then performed by analysing BB intersections in the hierarchy. Other techniques exploit proximity tracking [VM95] or curvature computation [Pro97] to reduce the large number of collision checks, excluding objects or parts which are impossible to collide.

Another approach to CD is based on image-space (IS) tests [SF91], [MOK95], [BWS99]. These algorithms use the graphics hardware to render the scene and then perform checks for interference between objects based on the depth information of the rendered image. In this way the 3D problem is reduced to 2.5D. Vassilev et al. [VSC01] applied this technique for detecting collisions between cloth and body when dressing virtual characters. They created depth, normal and velocity maps using two orthogonal cameras that were placed at the centre of the front and the back face of the body's BB. The depth map was used for detecting collisions, while the normal and velocity maps were used for collision response. Since they perform cloth simulation on the CPU they have to read back the frame buffers from the GPU which is time consuming.

Heidelberger at al [HTG04] extended the image space based approach to also deal with self-collisions by creating separate layered depth images (LDIs) on the GPU for front and back facing polygons. Their approach only works with water tight volumes and also requires the reading back of the frame buffer to the CPU for analysis.

Allard et al [AFC*10] recently built on the LDI approach but moved the whole simulation of deformable volumes to the GPU, avoiding the bottleneck of framebuffer readback. However, their approach also relies on watertight volume geometry.

In Govindaraju et al. [GKLM07] collision detection is regarded as a visibility problem and they use occlusion queries on the graphics processor to detect collisions at fast rates. Their performance tests show collision detection at rates over 100ms though.

Another approach to perform collision detection on the GPU was introduced by Sud et al [SGG*06]. They create a discrete Voronoi diagram (DVD) of the scene on the GPU and can therefore access proximity and collision information. This is useful if the topology of the geometry can change, as for example in fractures, etc. However, they report rates of several hundred milliseconds just for creating the discrete Voronoi diagrams.

The method described in this paper exploits the idea of the image-space approach of Vassilev et al. [VSC01] but we implement it entirely on the GPU including the cloth simulation and therefore we eliminate the bottleneck of frame buffer readback and the analysis of the framebuffer for collision detection and response on the CPU. Moreover, we extend the algorithm to test not only for cloth-body collisions, but also to cloth-cloth collisions. Unlike [HTG04] and [AFC*10] our self-collision tests are not based on LDIs but our method exploits the information we have on the GPU about the separate layers of cloth. Our method therefore does not rely on watertight volume geometry and we are able to perform collision detection and cloth-cloth collision detection at rates much higher than previously reported.

3. MASS-SPRING CLOTH ON THE GPU

Algorithm

Implementation of the mass-spring cloth model [VSC01] on the CPU requires an algorithm similar to the following pseudo code:

```
For each spring
  Compute internal forces
  Add forces to 2 end mass points
Endfor
For each mass point
  Add external forces
  Compute velocity
  Do collision detection and response
  Correct velocities for over-elongated
     springs
  Compute new positions
Endfor
```

Two implementations of the mass-spring system are possible on the GPU, which were compared by Georgii and Westermann [GW05]. The first one is to directly follow the CPU implementation, which they call edge-centred approach (ECA). The main difficulty here is to distribute the spring forces to the correct mass points with the correct sign. To solve this they use vertex shaders, but two additional render passes are required. The advantage of the ECA is that spring forces are computed only once, but it has several drawbacks:

- it requires more graphics memory for spring and force textures;
- it requires at least four rendering passes;
- additive blending in the render target is used to accumulate the force contributions, which has precision problems on some cards.

The second implementation uses only one for loop (for each mass point) and the computation of the spring forces is the first step inside. As a result it can be implemented in only one or two render passes, requires less graphics memory and is more straightforward to implement. Its only disadvantage is that each spring force is computed twice, but considering the parallel nature and tremendous power of recent GPUs this is negligible. Therefore the work in this paper is based on the second method. The texture units needed for our algorithm are described in the next sections.

GPU data structures for the cloth model

On the GPU the mass-spring model naturally maps into several texture2Ds. Several important facts have to be considered, when organising the data. If a texture is set as a rendering target, it is not available for reading. So, in order to compute the new velocities and positions of cloth vertices, the old values have to be stored in another texture, just for reading. We use the so called "ping-pong" technique [Göd07]: after a computational step, the two textures are swapped, and the texture from which was read before becomes the new rendering target.

Therefore, we need two textures (read/write) for velocities, two textures (read/write) for positions and one texture for normal vectors of the cloth surface at each cloth vertex, which are also computed on the GPU.

The main idea of this work is to store information about the springs connected to each mass point in an additional texture, which we call "spring connectivity texture". A suitable constraint on the maximum number of other vertices connected to a given mass point has to be imposed. For our cloth model this number is 12, owing to GPU architecture it is simpler to reserve 16 values, therefore we have 4 values for future extensions. If the textures for velocities, positions and

normals have size (texSize × texSize), then the spring connectivity texture is of size (4*texSize)×(4*texSize). This spring matrix consists of 16 smaller matrices. Each entry in these 16 matrices has 4 channels (RGBA) and keeps the following information of a spring connected to the corresponding vertex: texture coordinates of the other spring end point, natural length and spring stiffness. If all channels are equal to -1.0, this means that the entry represents no connection.

Seaming of garment pieces

Our Virtual Try-On system reads a body file and a garment file and dresses the body. The garment file holds information about the geometry of the cutting patterns, physics parameters of the cloth and seaming information. The patterns are automatically positioned around the body and external forces are applied along the seaming lines. The seaming lines are discretized into groups of two or three cloth vertices to be sewn together.

The connectivity of cloth vertices into seams is stored in a similar texture as for the springs, which we call "seam connectivity texture". Each entry of the texture keeps information about the other mass points to which the current cloth vertex has to be sewn. During the simulation forces are applied which pull together the corresponding vertices. When the vertices are closer than a certain threshold, they are marked as sewn and are attached to each other. The simulation ends when all seams are done, or after a pre-defined number of iterations which means the garment is too small for the body.

Occlusion queries for counting sewn vertices

In order to identify when the garment is sewn, the number of vertices still to be sewn have to be counted. Counting on the GPU can be performed using a reduction approach similar to the max reduction, described by Harris [Har05]. However, it cannot be applied directly. First, a 2D buffer has to be built, which contains ones for the not sewn vertices and zeros for the sewn ones. Then a sum reduction should be applied to the buffer, which will perform the required count.

In our system we utilize GPU occlusion queries for counting. Occlusion queries are implemented in hardware on most of the recent graphics cards and they allow the programmer to render a 3D scene and to obtain the number of fragments that passed the depth test. There is an internal counter, which is initially set to zero, and during the rendering it is increased by one for each fragment that passes the depth test.

In order to use occlusion query in our case, the following steps have to be carried out:

- Allocate a depth texture;
- Set this texture as a depth buffer for rendering;
- Render a suitable geometry and perform an occlusion query to retrieve the number of fragments that pass the depth test.

After the new positions were computed by the mass spring simulation we call a shader which holds the sewn vertices together and also builds a seam depth texture with the following values: 0 if the vertex is not involved in a seam, 1 if the vertex is part of a seam but is not sewn yet and 0.5 if the vertex is part of a seam, which is sewn. Then we set this texture as the default depth buffer. The z-buffer is turned to read only, otherwise the depth values will be replaced with the ones of the incoming vertices. Next we render a quad with a depth value of 0.8 that covers the whole draw buffer. In this way the occlusion query counts all fragments with a depth value greater than 0.8, that is the number of unsewn vertices. In fact we can render a quad with any depth value in the interval (0.5, 1).

The function for counting unsewn vertices does not need to be called after every integration step. To speed the simulation up it could be called after every 10 or 15 iterations.

Cloth Modelling Shaders

The following shaders are used in the system:

Velocity shader. This is the main cloth simulation shader. It computes the forces applied to each cloth vertex due to springs' tension, gravity, damping and seaming, then integrates over time to compute velocity and writes the result in the velocity texture. It also checks for collisions, as described below and if there is a collision it applies a force and also modifies the velocity to resolve the collision.

Position shader. It reads from the velocity texture and computes the new cloth vertex positions.

Seam shader. It checks if the cloth vertices that participate in a seaming line are close enough to be considered sewn, holds the sewn vertices together and builds the seam depth texture, as described in the above section.

4. COLLISION DETECTION BODY-CLOTH AND CLOTH-CLOTH

Cloth-body collision detection

As explained in Section 2 this paper exploits the idea of Vassilev et al. [VSC01] for collision detection. However, we do not build velocity maps, because the current system does not animate the virtual body representation. The garment is dressed and simulated on a static body. To build the normal maps more effi-

ciently we use a simple vertex shader, which replaces the vertex colour RGB values with the XYZ coordinates of the normal. In addition, to reduce the number of texture units, the front and back maps are placed in a single texture, as shown in Figure 1.

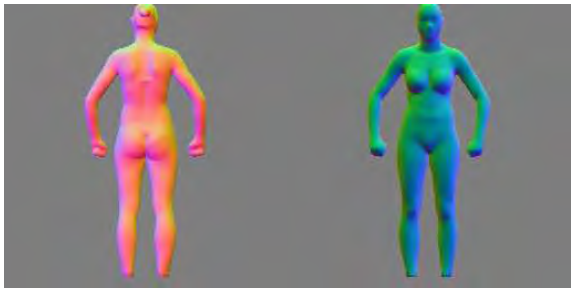


Figure 1: *Front and back normal maps in one texture*

And finally the normal and depth maps are placed in the same texture unit; RGB representing the normal coordinates and the alpha channel contains the depth. This speeds up the simulation, because when testing for collisions the velocity shader samples the collision texture only once to get depth and normal values of the front and the back of the body.

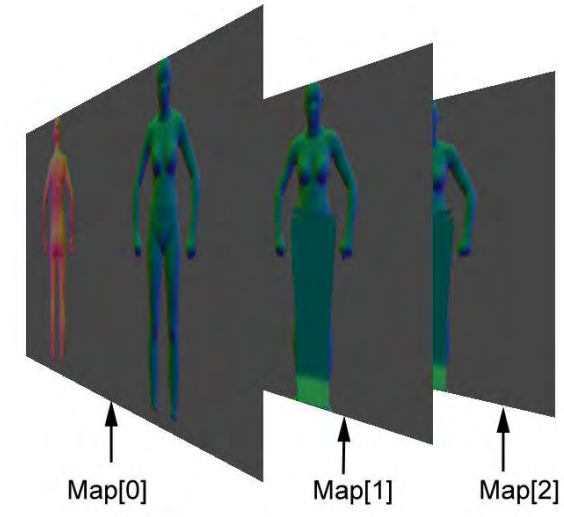


Figure 2: *Normal maps generated for collision detection*

Cloth-cloth collision detection

Our system does not aim at detecting all cloth-cloth collisions. It does not test for collisions in one piece of cloth as such are less likely to happen in tight garments on static body we simulate. When constructing garments some pieces of cloth have to be placed on top of others, for example pockets, belt loops, etc. When a person tries on two garments one is always on top of the other for example a T-shirt and a pair of jeans. The cloth pieces are grouped into layers, to know which layer is on top of other layers and we can assign a layer number to each cloth piece, starting from zero. Our system only tests for collisions between these different cloth layers.

The idea of this work is to use the same image-space approach for detecting collisions between layers of cloth. For this purpose several maps are generated (Fig. 2):

- Map[0]: body depth and normals
- Map[1]: body and cloth-layer 0 depth and normals
- …
- Map[*n*]: body and cloth-layers 0 to *n*-1 depth and normals

The number of maps, *n*, depends on the number of cloth layers we wish to simulate. Map[0] is generated only once at the beginning of the simulation, because the body does not move in our case. All other maps have to be generated at each iteration step. If we want to simulate garments on a body in motion, map[0] has to be generated at each iteration as well.

Collision checking

The computation of internal, external forces and velocities, as well as collision detection and response is performed in the velocity shader, as described above. This shader is called only once per integration step. When testing for collisions of a particular cloth vertex, which belongs to cloth layer *i*, we have to check if it collides with the body and all layers beneath it, which means that we have to use map[*i*] for CD and response. All maps are stored in a 3D texture in which each slice corresponds to a depth/normal map, as described above. The properties of each cloth vertex, such as mass, elasticity, cloth layer number, etc., are stored in another texture. The velocity shader uses the layer number to sample the appropriate slice of the 3D texture, for example cloth layer with number *i* samples slice *i* and uses it for collision detection and response as described in [VSC01]. The depth value of the current mass point is compared to the depth value read from the depth map. If a collision is detected a repulsive force is computed and applied to the cloth vertex using the normal vector, retrieved from the normal map. The velocity of the cloth vertex is also modified. The collision response enables us to simulate friction between layers, too.

Applying this approach allows us to simulate one-way interactions only. The lower layers push the upper layers back to prohibit penetration, but the upper layers do not affect the layers below. So, if a pair of jeans is dressed on top of a loose shirt, the jeans will not push the shirt towards the body.

In order to model interaction in both directions we do the following. The faces on the cloth surface are numbered from one to the number of faces and each number is encoded as vertex colour. When generating

the depth and normal maps the cloth surface is rendered using flat shading, so that these colours are not interpolated and the face colour encodes the face number. The fragment shader, used for the generation of the maps, stores the XYZ values of the normal vector in the RGB values of the map and the alpha value is computed as follows:

$$map\_alpha = depth + face\_number, \quad (3)$$

where face_number is decoded from the colour values. As the depth value is from 0 to 1 and the face number is greater than or equal to one, the two values can be easily separated. If a collision is detected the velocity shader, in addition to applying repulsive forces and modifying velocities, also writes the following alpha in the velocity texture:

$$velo\_alpha = face\_number. \quad (4)$$

If there is no collision the alpha is set to zero.

Another pair of vertex/fragment shaders is used to apply forces to lower layers of cloth. If a lower layer cloth face has collided with an upper layer vertex, we apply forces to each of the three vertices of this face, which are opposite to the face normal. These forces have to be summed up for each vertex, as a vertex can be part of several adjacent faces that have collided. In fact we integrate the forces over time and add them to the velocities. One of the velocity textures is set as a rendering target and the velocities are rendered three times as points with additive blending, once for each vertex of a triangular face. A uniform variable is set to 0, 1 or 2 before the rendering to define which face vertex is targeted. The vertex shader checks the forth coordinate of the velocity. If it is greater than zero, then this is a face which has collided to an upper layer cloth vertex. The indices of the cloth vertices of that face are read from an index texture. Knowing the size of the velocity texture and the index, the output position of the vertex shader is computed so that it projects to the targeted cloth vertex (number 0, 1 or 2 of the face) in the rendering target. As a result the fragment shader is executed for this cloth vertex and it applies a constant force opposite to the cloth normal, multiplied by the time step, and in this way pushes the cloth back. The magnitude of the force is determined experimentally. If there was no collision, the new position is computed so that it is outside the viewing volume and the fragment shader is not executed for this vertex.

Maps generation

As mentioned above map[0] is generated only once at the beginning. After each integration step we have to render maps from 1 to *n* in each slice of the 3D texture. In order to speed the simulation up, when generating map[i], we first copy map[i-1] to map[i], set it as the default colour and depth buffers and render cloth layer number (i-1). In this way the body and all cloth layers from 0 to (i-2) are already present in the frame buffer and do not have to be rendered again. So we have to render only the pieces with a layer number (i-1).

5. RESULTS

The system was implemented in OpenGL and GLSL. Figure 3 shows an example of the simulation of a pair of jeans and gives a closer look of pockets and belt loops.

In order to check the system performance, it was tested on several configurations. Two implementations of the Virtual-Try-On system were compared: 1) programs running on the CPU and 2) GPU-based as described in this paper. For the second one the textures used to store the cloth vertices positions and velocities were of a size 64 × 64. The influence of the upper to lower layers was not simulated, as for a single garment it is not significant. The performance results of 3 GPUs and the fastest CPU are given in Table 1 and Figure 4. They show that the virtual try-on runs very well on a modern laptop GPU, which is about 20 times faster than the fastest tested CPU. One iteration includes integration, collision detection and response.

Figure 3: Layers of cloth: pockets and belt loops

The average time of putting a garment on a virtual body using the NVIDIA GTX 460 GPU is about 0.2 seconds depending on the garment complexity and size.

Figure 6 shows results of the simulation of two garments, jeans dressed on top of a shirt. The jeans were discretised with 3600 mass points and 2100 vertexes were used for the shirt. Two modifications of the cloth-cloth collision detection algorithm are depicted:

left – no impact of upper to lower layers. The simulation is faster (0.20 sec for dressing the jeans), but not satisfactory.

right – with impact of upper to lower layers. The simulation is slower (0.34 sec for dressing the jeans), but of much better quality. The time spent only on collision detection and response is 0.21 seconds for the whole cycle of dressing a pair of jeans, which requires 675 iterations. This means that for each iteration 0.31 ms is spent on cloth-cloth collision detection and response.

	Time for 1000 iterations, s	Iterations per second
Intel core i7, 2.2 GHz	8.65	116
ATI Radeon HD4850	0.57	1769
NVIDIA GeForce GTX 560M laptop	0.40	2491
NVIDIA GeForce GTX 460	0.30	3300

Table 1: **Performance of the system, measured on 3 GPUs and a CPU**

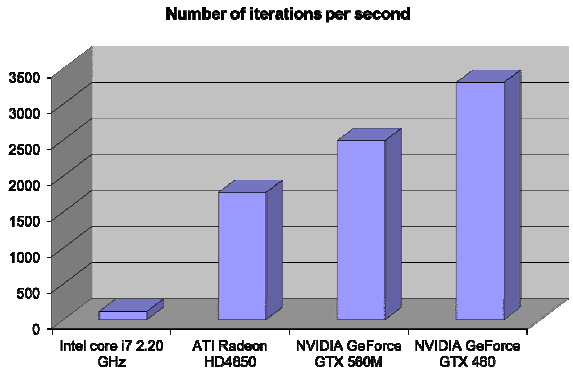


Figure 4: Performance of CPU and 3 GPUs

The algorithms for cloth simulation and collision detection and response were also implemented using NVidia CUDA and OpenCL. The comparison with the GLSL implementation [Vas10] showed that GLSL outperforms CUDA (OpenCL). One of the main reasons is that CUDA (OpenCL) and OpenGL have to share buffers for the rendering and this buffers should be mapped and later unmapped when used in CUDA (OpenCL), which slows down the simulation.

Figure 6: **Jeans dressed on a shirt; left: no impact of upper to lower layers; right: with impact of upper to lower layers**

6. CONCLUSIONS

An efficient technique for dynamic garment simulation entirely on the GPU has been presented. It implements a mass-spring system with velocity modification to overcome super elasticity and exploits an image-space approach for collision detection and response. The following more important conclusions can be drawn:

- A general mass-spring system can be implemented on the GPU using several textures for storing data and several connectivity textures for keeping spring and seaming information.
- Hardware assisted occlusion queries can be utilised for counting unsewn cloth vertices, which speeds simulation up.
- The same image-space based approach can be applied for detecting collisions cloth-body and cloth-cloth when layers of cloth are simulated. Multiple collision maps can be stored in a 3D texture.

The system can simulate approximately 20 garments per second on a PC with 2 dual GPU NVidia GeForce graphics cards. This is currently sufficient for our web based Virtual Try On service.

The system can be extended to simulating garments on animated virtual characters. For this purpose velocity maps will also have to be generated and stored in another 3D texture.

7. ACKNOWLEDGEMENTS

Tzvetomir Vassilev's work is partly supported by a National Research Fund project at the University of

Ruse, Bulgaria. Bernhard Spanlang's work is partially supported by the ERC project TRAVERSE.

8. REFERENCES

[AFC*10] Allard J., Faure F., Courtecuisse H., Falipou F., Duriez C., Kry P. G.: Volume contact constraints at arbitrary resolution. ACM Trans. Graph. 29, 4 (2010), 1–10.

[BHW94] Breen D., House D., Wozny M.: Predicting the drape of woven cloth using interacting particles. In Computer Graphics Proceedings, Annual Conference Series (1994), vol. 94, pp. 365–372.

[BW98] Baraff D., Witkin A.: Large steps in cloth simulation. In Computer Graphics Proceedings, Annual Conference Series (1998), SIGGRAPH, pp. 43–54.

[BWS99] Baciu G., Wong W. S., Sun H.: Recode: an image-based collision detection algorithm. The Journal of Visualization and Computer Animation 10, 4 (1999), 181–192.

[CYTT92] Carignan M., Yang Y., Thalmann N. M., Thalmann D.: Dressing animated synthetic actors with complex deformable clothes. In Computer Graphics Proceedings, Annual Conference Series (1992), vol. 92, pp. 99–104.

[EWS96] Eberhardt B., Weber A., Strasser W.: A fast, flexible, particle-system model for cloth draping. j-IEEE-CGA 16, 5 (Sept. 1996), 52–59.

[GKLM07] Govindaraju N. K., Kabul I., Lin M. C., Manocha D.: Fast continuous collision detection among deformable models using graphics processors. Comput. Graph. 31, 1 (2007), 5–14.

[Gla98] Glassner N. I. B. A. S.: 3d object modelling. SIGGRAPH 12, 4 (1998), 1–14.

[Göd07] Göddeke D.: Gpgpu::basic math tutorial, 2007.

[GW05] Georgii J., Westermann R.: Mass-spring systems on the gpu. Simulation Modelling Practice and Theory 13 (2005), 693–702.

[Har05] Harris M.: Mapping computational concepts to gpus. In SIGGRAPH '05: ACM SIGGRAPH 2005 Courses (New York, NY, USA, 2005), ACM, p. 50.

[HTG04] Heidelberger B., Teschner M., Gross M.: Detection of collisions and self-collisions using image-space techniques. In Journal of WSCG (2004), pp. 145–152.

[LHG*06] Luebke D. P., Harris M., Govindaraju N. K., Lefohn A. E., Houston M., Owens J. D., Segal M., Papakipos M., Buck I.: S07 - gpgpu: general-purpose computation on graphics hardware. In SC (2006), ACM Press, p. 208.

[MHS05] Mosegaard J., Herborg P., Sørensen T. S.: A gpu accelerated spring mass system for surgical simulation. Studies in health technology and informatics 111 (2005), 342–348.

[MOK95] Myszkowski K., Okunev O. G., Kunii T. L.: Fast collision detection between complex solids using rasterizing graphics hardware. The Visual Computer 11, 9 (1995), 497–512.

[Pro95] Provot X.: Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In Proceedings of Graphics Interface (1995), pp. 141–155.

[Pro97] Provot X.: Collision and self-collision detection handling in cloth model dedicated to design garments. In Proceedings of Graphics Interface (1997), pp. 177–189.

[PTVF92] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: Numerical Recipes in C, 2nd. edition. Cambridge University Press, 1992.

[RNS06] Rodriguez-Navarro X., Susín A.: Non structured meshes for cloth gpu simulation using fem. In 3rd. Workshop in Virtual Reality, Interactions, and Physical Simulations (VRIPHYS) (2006), pp. 1–7.

[RNSS05] Rodriguez-Navarro X., Sainz M., Susin A.: Gpu based cloth simulation with moving humanoids. In Actas XV Congreso Español de Informática Gráfica (CEIG'2005) (2005), J. Regincós D. M., Thomson-Paraninfo E., (Eds.), pp. 147–155.

[SF91] Shinya M., Forgue M. C.: Interference detection through rasterization. j-J-VIS-COMP-ANIMATION 2, 4 (Oct.–Dec. 1991), 132–134.

[SGG*06] Sud A., Govindaraju N., Gayle R., Kabul I., Manocha D.: Fast proximity computation among deformable models using discrete voronoi diagrams. ACM Trans. Graph. 25, 3 (2006), 1144–1153.

[TPBF87] Terzopoulos D., Platt J., Barr A., Fleischer K.: Elastically deformable models. Computer Graphics (Proc. SIGGRAPH'87) 21, 4 (1987), 205–214.

[Vas10] Vassilev T.I.: Comparison of several parallel API for cloth modelling on modern GPUs. In Proceedings of CompSysTech (2010).

[VM95] Volino P., Magnenat Thalmann N.: Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In Computer Animation and Simulation '95 (1995), Terzopoulos D., Thalmann D., (Eds.), Springer-Verlag, pp. 55–65.

[VSC01] Vassilev T., Spanlang B., Chrysanthou Y.: Fast cloth animation on walking avatars. Computer Graphics Forum 20, 3 (2001), 260–267. ISSN 1067-7055.

[Zel05] Zeller C.: Cloth simulation on the gpu. In SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches (New York, NY, USA, 2005), ACM, p. 39

Improving Active Learning with Sharp Data Reduction

Priscila T. M. Saito[†], Pedro J. de Rezende[†], Alexandre X. Falcão[†],
Celso T. N. Suzuki[†], Jancarlo F. Gomes[†‡]
[†]Institute of Computing, University of Campinas - UNICAMP, Campinas, SP, Brazil
[‡]Institute of Biology, University of Campinas - UNICAMP, Campinas, SP, Brazil
{maeda, rezende, afalcao, celso.suzuki, jgomes}@ic.unicamp.br

ABSTRACT

Statistical analysis and pattern recognition have become a daunting endeavour in face of the enormous amount of information in datasets that have continually been made available. In view of the infeasibility of complete manual annotation, one seeks active learning methods for data organization, selection and prioritization that could help the user to label the samples. These methods, however, classify and reorganize the entire dataset at each iteration, and as the datasets grow, they become blatantly inefficient from the user's point of view. In this work, we propose an active learning paradigm which considerably reduces the non-annotated dataset into a small set of relevant samples for learning. During active learning, random samples are selected from this small learning set and the user annotates only the misclassified ones. A training set with new labelled samples increases at each iteration and improves the classifier for the next one. When the user is satisfied, the classifier can be used to annotate the rest of the dataset. To illustrate the effectiveness of this paradigm, we developed an instance based on the optimum path forest (OPF) classifier, while relying on clustering and classification for the learning process. By using this method, we were able to iteratively generate classifiers that improve quickly, to require few iterations, and to attain high accuracy while keeping user involvement to a minimum. We also show that the method provides better accuracies on unseen test sets with less user involvement than a baseline approach based on the OPF classifier and random selection of training samples from the entire dataset.

Keywords: Pattern Recognition, Machine Learning, Active Learning, Semi-Automatic Dataset Annotation, Data Mining, Optimum-Path Forest Classifiers.

1 INTRODUCTION

The amount of available information has been increasing due to the advances of computing and data acquisition technologies, resulting in large datasets. Handling and analysing such increasing volume of information have become humanly infeasible and highly susceptible to errors, since it is extremely time consuming and wearisome. Hence, there is an increasing demand for the development of effective and efficient ways to annotate these datasets.

Active learning techniques have been explored and reasonably successful. However, these methods fall in a single paradigm which requires, at each iteration, the classification of the entire dataset under annotation, followed by the organization of all these samples according to some criterion, in order to select the most informative samples to be used for training the classifier. These phases are highly interdependent and, for large datasets, performing them at each iteration is very inefficient or even computationally infeasible.

In this paper, to overcome the aforementioned problems, we propose a new active learning paradigm which is verifiably effective and more efficient in practice, when dealing with large datasets, than those based on the current state of the art. The proposed paradigm relies on a significant reduction in the dataset size to create a small representative set of samples, for the learning process. By constructing the first instance of the classifier based on the knowledge of as many classes as possible, as well as incorporating the best samples at each iteration, subsequent selection and classification phases are much more efficacious. This approach differs from the traditional active learning methods, in which *all* samples in the database have to be classified and re-organized at each iteration.

Being a paradigm, it can be implemented using different strategies. This paper also presents an instantiation (Cluster-OPF-Rand) which has been developed to illustrate the effectiveness of this paradigm. It is based on the Optimum Path Forest (OPF) classifier, while relying on clustering and classification for the learning process. Cluster-OPF-Rand prevents the user from having to annotate a large (and usually wasteful) number of training samples. Moreover, it prevents poor selection of sam-

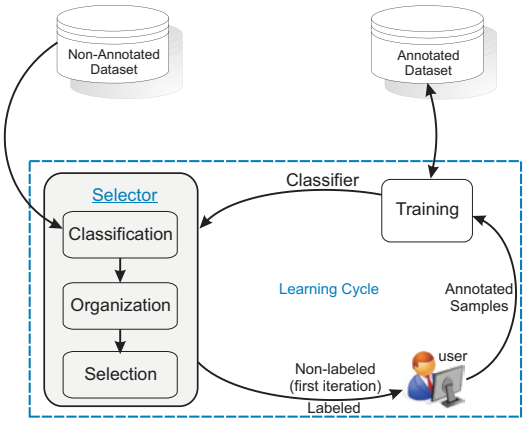


Figure 1: Pipeline of the traditional active learning paradigm.

ples from a large learning set, since this set is reduced so as to contain essentially the most representative samples. After this reduction, the proposed paradigm enables the organization of the learning samples to occur beforehand (and only once). In this particular implementation, the organization of the reduced set occurs in a randomized fashion.

The experiments performed on three datasets show that Cluster-OPF-Rand is interactively and iteratively efficient, in addition to providing high accuracies earlier. That is to say, the number of learning iterations is significantly reduced with better accuracies, while requiring the annotation of only a small number of samples, when compared to a baseline approach using the OPF classifier and random selection of training samples from the entire dataset. The results also showed impressive reductions of over 90% in user effort, at the same time providing accuracies of over 97%.

The remainder of this paper is structured as follows. Section 2 summarizes the major active learning techniques presented in the literature. Section 3 presents the clustering approach based on optimum-path forest used. Section 4 details the active learning paradigm and the reduced method proposed. Section 5 discusses the experiments and the accomplished results. Finally, Section 6 presents the conclusions and future work.

2 BACKGROUND AND TECHNIQUES

Recent works in active learning have yielded a variety of heuristics, which are designed mostly for binary classification and are applicable primarily to classifiers such as Artificial Neural Network (ANN), Support Vector Machine (SVM), k-Nearest Neighbour (k-NN) and Optimum-Path Forest (OPF).

In active learning techniques, the key idea relies on the strategy used to select the most informative samples such that they allow for the achievement of greater accuracies with fewer training labels annotated by the user. Much effort has been placed in investigating strategies for active learning. However, it is focused on methods that classify all samples in the database, then organize these samples according to certain criteria and subsequently select and display the most informative samples to be annotated by the user, at each learning iteration. For large databases, these complete phases, at each learning iteration, are very inefficient or even impractical to be done computationally.

Figure 1 illustrates the execution pipeline of the traditional active learning paradigm presented in prior literature. This paradigm is comprised of a learning algorithm and a selector. The selector consists of three modules (classification, organization and selection) that are highly interdependent. At each iteration cycle, the system presents to the user a set of samples that consists of either non-labelled samples (from the entire database, in the first iteration) or labelled ones (obtained through the classifier), all chosen by the selector. As these samples are annotated by the user, they are included in the training set to retrain the classifier for the next cycle.

Besides the aforementioned inefficiency, most of the existing research in the traditional active learning paradigm has focused on binary classification. Relatively few approaches [12, 20, 9, 16, 11, 10] have been proposed for multiclass active learning and are typically based on extensions of predominantly binary active learning methods to the multiclass scenario.

In the ANN literature, although several works [4, 1, 7] have explored the use of active learning in the context of efficient network training, this approach shows the disadvantage of being computationally expensive.

Alternatively, SVM has been used in [19, 18], under the assumption that the samples closest to the separating hyperplane are the most informative ones. During the iterations of relevance feedback, the method finds the optimal hyperplane separating relevant and irrelevant samples and presents to the user the samples closest to this hyperplane. This hyperplane is adjusted throughout the iterations, and after the last one, the method presents the most relevant samples as being the farthest ones to

the hyperplane. Extensions to the multiclass scenario are typically based on extensions of binary classification using pairwise comparisons or 1-vs-all strategy.

In contrast, [10] introduced a probabilistic variant of k-NN. Although, this variant was designed specifically for multiclass problems, it involves learning a certain number of parameters. Moreover, the performance of the method is dependent on the similarity measure used.

A strategy, similar to the one presented in [18], was proposed in [6], using a faster and more effective classifier based on Optimum-Path Forest (OPF). They developed greedy (GOPF) [5] and planned (POPF) [6] active learning strategies for CBIR systems. For a given set of relevant and irrelevant samples, the method computes an optimum-path forest using samples from the query set for training the classifier.

Optimum-Path Forest (OPF) is a framework for developing pattern classifiers (supervised, semi-supervised or unsupervised) which defines how the samples are connected by an adjacency relation that gives rise to a graph, and how to measure the connectivity (the cost of a path in the graph generated by the adjacency) between them by means of a function that gives rise to an optimum path forest.

The supervised and the unsupervised classifiers were described in [14, 17], respectively. Both learning approaches are fast and robust for large datasets [13, 2]. In addition, the classes/clusters may present arbitrary shapes and have some degree of overlapping. Classifiers based on OPF have been widely used in several applications and have demonstrated that OPF-based classifiers can be more effective and much faster than ANN and SVM based ones [14].

The following Section details the OPF based clustering approach.

3 CLUSTERING BY OPTIMUM-PATH FOREST

The data reduction approach we implemented is based on clustering by Optimum-Path Forest (OPF) [17]. This is a non-parametric approach which estimates the number of natural groups in a dataset as the number of maxima of its probability density function (pdf). In this approach, each maximum of the pdf will define a cluster as an optimum-path tree rooted at that maximum. It can handle plateaux of maximum, by electing a single root (one prototype per maximum), groups with arbitrary shapes, and some overlapping among clusters.

In this unsupervised learning algorithm, an unlabelled training set is interpreted as a graph whose nodes are samples (images, in this paper) and each node is connected with its k-closest neighbours in the feature space to form directed arcs. The pdf value at each node is estimated from the distance between adjacent samples, and a connectivity (path-cost) function is designed such that the maximization of a connectivity map defines an optimum-path forest rooted at the maxima of the pdf. In this forest, each cluster is one optimum-path tree rooted at one maximum (prototype). The pdf estimation also requires multiple applications of the algorithm for different values of k in order to select the best clustering result as the one that produces a minimum normalized cut in the k-NN graph. The clusters are found by ordered label propagation from each maximum, as opposed to the mean-shift algorithm of [3] which searches for the closest maximum by following the direction of the gradient of the pdf — a strategy that does not guarantee the assignment of a single label per maximum, and presents problems on the plateaux of the pdf.

In order to handle large datasets, this approach estimates the pdf from random samples and fast propagates the group labels to the remaining samples of the dataset. The best k for pdf estimation is found by optimization, but its search interval [1, kmax] may produce different numbers of groups. The parameter kmax represents an observation scale for the dataset. If kmax is too high, it means that we are looking at the dataset from infinity and so, the result will be a single cluster. As we approximate the dataset (reducing the value of kmax), the number of clusters increases up to some high number for kmax = 1. Still, the number of possible solutions is low, because the method produces an identical number of clusters for several values of kmax. This shows the robustness of the method in finding natural groups in the dataset for distinct observation scales. In this work, we chose kmax so as to obtain a number of groups higher than the number of classes known. Note that, we do not use any knowledge on the classes of samples, but we assume that we know how many classes are present in the dataset.

4 PROPOSED PARADIGM

We propose a new paradigm for active learning in order to select, more efficiently and effectively, a small number of the most representative samples for training a classifier. The execution pipeline of the proposed paradigm is illustrated in Figure 2.

In the proposed paradigm, a classifier instance is generated at each iteration. After retraining the classifier (a process that relies on user annotations), the selector displays the most informative samples to the user. As the classifier improves, the user is required to correct fewer misclassified samples and progressively develops a sense of when the learning process has reached a satisfactory state.

Active learning methods presented in the literature differ from one another in their learning algorithms and in the selection strategies employed. The main difference

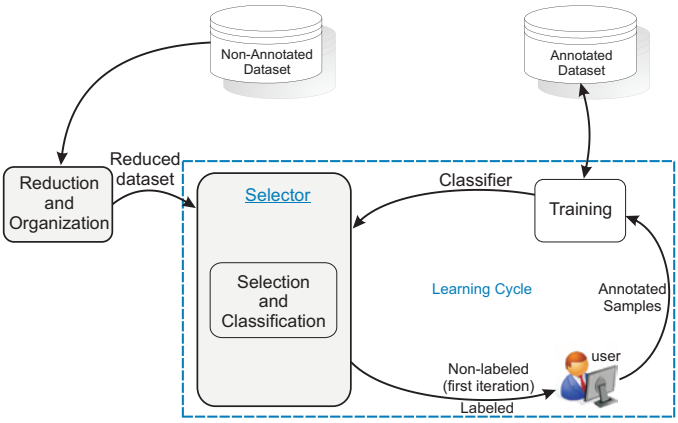


Figure 2: Pipeline of the proposed active learning paradigm.

between the proposed paradigm and previously proposed ones lies within the selector. Traditional methods make use of three modules that correspond to classification, organization and selection of samples (Figure 1). In these methods, the selection criterion is based solely on a classifier that is not yet reliable. When the classification accuracy is still low, the organization phase becomes useless, since when samples are classified, informative samples may not be selected to participate in the organization phase and therefore they will not be shown to the user.

The proposed paradigm is based on a priori data reduction and organization of the reduced dataset. It focuses on reversing the process adopted by traditional paradigms where an classification phase occurs before the organization phase. In the proposed paradigm, the selector consists of only one module of selection and classification. A major advantage presented by the proposed paradigm is that the reduction and organization of samples can be performed only once, unlike traditional methods.

Thus, the selector becomes faster, especially considering large databases, since the improvement of the classifier at each iteration does not require rearranging all samples; only the selection and classification phases are required. Moreover, a remarkably faster selection phase is completed by the choice of a small subset of samples and the classification of only these.

The strategy to be developed in order to select the most informative samples itself occurs as preprocessing in the module of reduction and organization (Figure 2). This strategy should not be based on a classifier, because it is still unreliable, but rather based on an absolute criterion previously established (for instance, exploring the organization of the data in the feature space). The classification phase is performed a posteriori, supporting the choice of the most informative samples by the selector, which follows a predetermined order in the reduction and organization module. In this module, different methods can be applied in our paradigm. In Subsection 4.1, we develop and present an effective method for the learning process.

4.1 Instantiation of the proposed paradigm

As it was mentioned, any method can be incorporated into the proposed paradigm in order to reduce the learning set and later to organize the reduced one. In this section, we present an effective method called Cluster-OPF-Rand. Figure 3 illustrates an example of the pipeline of Cluster-OPF-Rand.

The proposed method is divided into two modules: (1) reduction and organization, (2) selection and classification. The reduction and organization module is comprised of two steps: clustering and reduction of the data (steps 1 and 2 of Figure 3, respectively). The selection and classification module choose and label (steps 3 and 4 of Figure 3, respectively) the most informative samples of the reduced set chosen in a randomized fashion. Each sample is represented by a pair (*id*, *lbl*), where *id* corresponds to the identifier of the sample and *lbl* corresponds to the label given by the classifier. Note that it does not classify *all* samples in the dataset, but only the selected subset.

Initially, clusters are computed in order to obtain samples of all classes, as described in Section 3. One or more clusters represent samples of all classes in the non-labelled set, so that each cluster comprises mostly samples of a single class. Then, their roots (highlighted after step 1) cover samples of all classes and are defined as an initial training set for manual annotation. This is fundamental to be able to train the classifier with samples of all classes, since the first iteration. This classifier should be as good as possible because it is used in the classification of samples, providing an initial labelling, in which the user is not required to annotate all samples shown but only to correct a small number of misclassified ones.

Besides knowing which samples are roots of clusters, it is possible to identify those that are boundary sam-

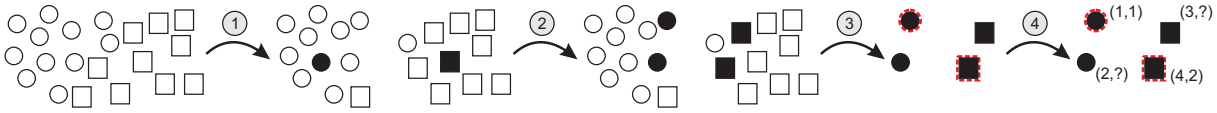


Figure 3: An example of pipeline of the proposed method.

ples between different clusters. A sample s is considered a boundary sample if there exists, among its k-NN adjacent samples, at least one whose label (given by the clustering) is different from that of s. The cluster boundary samples are expected to correspond to the boundary between classes. This identification of boundary samples allows for the reduction of the learning set to a small relevant set (consisting of boundary samples), since these can be considered as the most representative samples for improving the classifier.

In the first iteration of the learning phase, the roots of the clusters are displayed to the user, who annotates their labels. These samples constitute the training set for the first instance of the classifier. For all other iterations, among the samples of the reduced set (boundary samples of the clusters) a few randomly chosen ones are selected for classification. Once classified, these samples are submitted to the user for confirmation of the labels assigned by the current classifier. Since only a small number of misclassified samples require annotation, the user's time and effort are lessened. In fact, as the classifier improves throughout the iterations the actions required from the user are increasingly reduced. After the labels are confirmed/corrected by the user, the samples are incorporated into the training set and a new instance of the classifier is generated. This entire cycle is repeated until the user is pleased with the accuracy of the classification.

Moreover, it is important to emphasize that different clustering techniques (such as k-means or k-medoids) can be used in the data reduction phase. Similarly, different supervised classifiers can be used in the classification and selection phases of the proposed paradigm. We choose OPF-based clustering since it offers many advantages, as mentioned in Section 2.

5 EXPERIMENTS

For evaluation, we developed a baseline approach (OPF-Rand) using the OPF-classifier and random selection of samples. At each learning iteration, the same number of random samples is selected from the entire dataset for OPF-Rand and, from the reduced dataset, for the Cluster-OPF-Rand. This number of samples is equal to the number suggested by Cluster-OPF-Rand based on the clustering results – a fair choice. These samples are classified and presented to the user for annotation. The user annotates the misclassified samples and they are added to the training set to improve the OPF classifiers used in each method for the next iteration. Thus, one can easily note the gain obtained by using clustering for dataset reduction, which induces the knowledge of a large number of classes, resulting in an early increase in accuracy. Moreover, clustering also allows for the choice of random samples from the reduced set comprised of good representative samples, instead of a much larger set of data (as in OPF-Rand).

The reported results were compiled from the average of experiments run 10 times, with randomly generated learning sets and unseen test sets for accuracy measures. For all datasets used, we chose 80% of the available samples for learning, and 20% for testing.

5.1 The Dataset Description

To perform the experiments we have used real-world datasets from very diverse domains. Due to space limitations, in the present paper there are only results obtained from three datasets.

The first dataset was obtained from the University of Notre Dame [8]. It was originally designed to study the effect of time on face recognition. The images were acquired in several weekly sessions with the participation of distinct individuals. In these sessions, different expressions (neutral, smiling, sad) were captured. In this work, we concentrated on a subset containing 1,864 samples with 162 features and 54 classes. Figure 4 displays specimens from this dataset.

Figure 4: Examples of images from the Faces dataset.

The second dataset is composed of images of parasites, provided by a research laboratory at the University of Campinas, where faecal parasitological examination is performed for diagnosis of enteroparasitosis present in humans. We used a dataset consisting of 1,660 faecal samples with 262 features and 15 classes. A particularity of this set is that each class contains a different number of images varying from 33 to 163 depending on the parasite species found on microscope slides. Figure 5 displays samples from this dataset.

The third one is the Pen-Based Recognition of Handwritten Digits dataset obtained from the UCI Machine

Faces	Accuracy (%)		Total Annotated Images (%)	
Iteration	Cluster-OPF-Rand	OPF-Rand	Cluster-OPF-Rand	OPF-Rand
1	94.85	85.11	6.51	6.51
2	97.27	94.21	7.59	8.51
3	98.06	97.35	8.11	9.40
4	98.57	98.35	8.41	9.78
5	98.85	98.78	8.68	9.98

Table 1: Accuracies and total annotated images for Cluster-OPF-Rand and OPF-Rand on the Faces dataset.

Figure 5: Examples of images from each class of the structures of intestinal parasites in the Parasites dataset.

Learning Repository [15], that consists of 10,992 objects in 16 dimensions, distributed in 10 classes corresponding to the digits [0...9]. The 16 dimensions are drawn by re-sampling from handwritten digits. This digits database was built from a collection of 250 samples from 44 writers.

5.2 Results

To compare the effectiveness of each method (Cluster-OPF-Rand and OPF-Rand), Tables 1-3 present the mean accuracy and the total annotated images using the datasets Faces, Parasites and Pendigits, respectively. It is important to emphasize that comparisons were not performed between Cluster-OPF-Rand and methods that require classifying and organizing *all* samples in the database, at each learning iteration, due to this process being infeasible in practice.

Notice that the proposed method creates a new classifier instance at each iteration. We would like to verify the ability of Cluster-OPF-Rand in choosing the most representative samples from a reduced set, as well as, in which iteration, whether the user might be pleased with the classification accuracy. Therefore, we monitor the mean accuracy of each instance on the unseen samples of the test set. Furthermore, for each sample set selected at each iteration, we simulate the user interaction by correcting the misclassified labels given by the current classifier instance. Tables 1-3 help compare the total number of annotated images used to increase the training set.

In summary, Cluster-OPF-Rand started off with a better performance than OPF-Rand, for all datasets analysed. Moreover, it achieves high accuracies sooner. To reach the same accuracies, the randomized method (OPF-Rand) required more samples annotated by the user as well as more learning iterations than Cluster-OPF-Rand.

Using the Faces dataset (Table 1), both methods achieve similar accuracies and both can be improved with more user annotations and more learning iterations. However, Cluster-OPF-Rand allows the learning process to stop earlier in comparison with OPF-Rand. Furthermore, it is important to highlight that, out of 1,469 samples only 132.94 (about 9.05%) had to be annotated for the proposed method to achieve accuracy above 99%, in its last (9^{th}) iteration using all samples on the reduced set. These results are similar to those for the remaining datasets (Tables 2 and 3). This shows that our method can outperform OPF-Rand in effectiveness.

Considering the Parasites dataset (Table 2), in the first iteration, Cluster-OPF-Rand achieves accuracies above 92% with less than 2% of the learning samples annotated by the user, while the randomized method OPF-Rand reaches similar accuracies only from the fourth iteration on and requiring the user to annotate more than 3% of the learning samples. Furthermore, out of 1,323 samples only 77.7 (about 5.87%) had to be annotated for Cluster-OPF-Rand to achieve an accuracy above 97%, in its last (25^{th}) iteration using all samples in the reduced set.

For the Pendigits dataset (Table 3), our method obtains high accuracies in all learning iterations. In the first one, it presents an accuracy of 88.80%. In the remaining iterations, the accuracies tend to increase continuously, reaching over 99%. Furthermore, out of 8,791 samples only 79.9 (about 0.90%) had to be annotated for the proposed method to achieve accuracy above 97% in the 30^{th} iteration. In a practical situation, a user would be very pleased at this point, mainly considering that the randomized method (OPF-Rand) learning process consists of 440 iterations, when using all available learning samples.

Figure 6a-b illustrates the mean accuracies and the number of samples annotated by the user at each iteration for each dataset using Cluster-OPF-Rand,

Parasites	Accuracy (%)		Total Annotated Images (%)	
Iteration	Cluster-OPF-Rand	OPF-Rand	Cluster-OPF-Rand	OPF-Rand
1	92.68	79.44	1.98	1.98
2	94.12	88.50	2.54	2.66
3	94.94	91.60	2.91	3.06
4	95.30	92.67	3.12	3.29
5	95.21	93.64	3.36	3.54

Table 2: Accuracies and total annotated images for Cluster-OPF-Rand and OPF-Rand on the Parasites dataset.

Pendigits	Accuracy (%)		Total Annotated Images (%)	
Iteration	Cluster-OPF-Rand	OPF-Rand	Cluster-OPF-Rand	OPF-Rand
1	88.80	70.36	0.13	0.13
2	90.96	82.97	0.22	0.25
3	91.99	87.49	0.29	0.30
4	92.89	89.72	0.35	0.35
5	93.70	91.25	0.40	0.40

Table 3: Accuracies and total annotated images for Cluster-OPF-Rand and OPF-Rand on the Pendigits dataset.

respectively. We used logarithmic scales, due to the size of these datasets. Our method requires a greater effort by the user in the first few iterations, since the selected samples are the most difficult to classify. However, looking at the end of the learning phase, one can observe that the proposed method demands less effort from the user, who annotates much fewer samples after some iterations (reaching almost no annotations at all).

The reduction strategy becomes very important in a process where a goal is to limit the number of iterations to as few as possible. In this context, selecting samples that speed up the improvement of the classifier through the iterations becomes critical. The more difficult to classify the selected samples in the current iteration are, the more useful they are to improve the classifier for the next iteration. Therefore, the selection of hard to classify samples coupled with the early knowledge of all classes allow for higher accuracy sooner.

Note that, in the first iteration with all datasets (Tables 1-3), Cluster-OPF-Rand provides higher accuracies than OPF-Rand. Using roots of each cluster for the first classifier instance becomes really important due to its use in the next iteration. This reduces the time and effort by the user who mainly has only to confirm the labels of the samples that have already been classified. Hence, this first instance of the classifier should be based on the knowledge of as many classes as possible (ideally, all of them). In later learning iterations, the performance gain depends on the choice of good samples. With the proposed method, it is possible to improve these choices by reducing a large dataset to a small subset consisting of boundary cluster samples for the training of the subsequent classifiers.

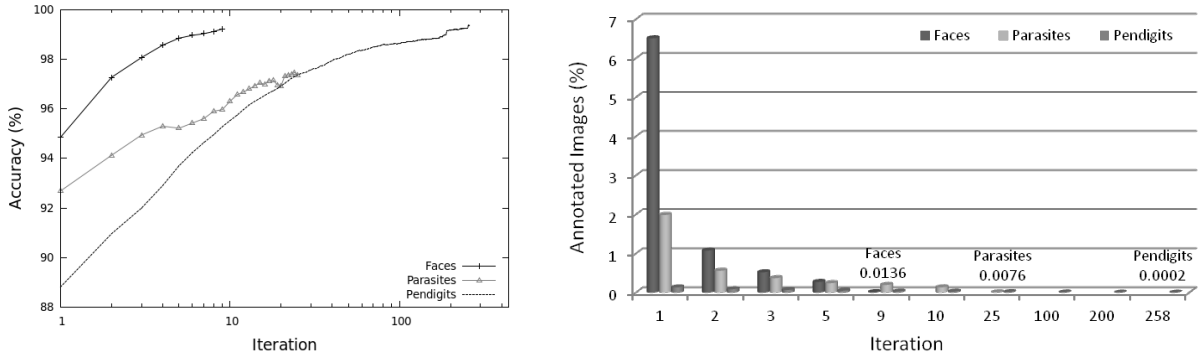
It is clear that Cluster-OPF-Rand, in addition to providing high accuracies, requires fewer learning iterations than those demanded by OPF-Rand. Additionally, it relies on fewer interactions with the user whose effort is reduced to almost none after a few iterations. Therefore, clustering improves the knowledge of samples from most/all classes. From the results presented, we can see that clustering roots allow us to obtain high accuracy since the first iteration. In the remaining iterations, the growth of accuracy is faster for Cluster-OPF-Rand, which also proves beneficial for the reduction strategy proposed.

6 CONCLUSION AND FUTURE WORK

In this work, we introduced an efficient active learning paradigm which enables the reduction and organization of the learning set a priori. A first instantiation, Cluster-OPF-Rand, of the proposed paradigm was developed in order to illustrate its effectiveness. The data reduction is based on clustering and the organization uses a randomized choice of samples of the reduced set, which contains the most representative (boundary) ones for the learning process. Cluster-OPF-Rand enables us to achieve the desired results, by using the knowledge of both user and classifier, at each learning iteration, along with the reduction strategy developed.

We concluded that our paradigm is more suitable to handle large datasets than the traditional one where methods require, at each learning iteration, the classification of *all* samples in the database, followed by their organization, and, finally selection. The proposed paradigm enables the reduction and organization phases to occur only once, as pre-processing. In addition, classification does not occur for all samples in the database, but to a small set of samples.

Experiments with datasets from distinct applications showed that Cluster-OPF-Rand, in addition to achiev-



(a) (b)

Figure 6: Comparison of Cluster-OPF-Rand on the three datasets. (a) Mean accuracy on the test sets. (b) Total annotated samples in each iteration (in percentage).

ing higher accuracy sooner, requires fewer learning iterations than those presented by OPF-Rand. Moreover, it is important to highlight that the user's time and effort are reduced to almost none after just a few iterations. Furthermore, experiments also demonstrated that it is possible to reduce the user's effort by over 90%, obtaining a classification accuracy above 97%.

Considering that new technologies have provided large datasets for many applications and that he traditional paradigms for active learning present unacceptable training times, we conclude that the proposed paradigm is an important contribution to active machine learning. Future works include developing other ways to explore the reduction and organization of data, for instance, a strategy that relies on an absolute criterion established a priori which explores the organization of the data in the feature space.

7 ACKNOWLEDGEMENTS

This work has been supported by grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq): 481556/2009-5, 303673/2010-9, 552559/2010-5, 483177/2009-1, 473867/2010-9; from Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES): 01-P-01965/2012; from Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP): 07/52015-0 and from FAEPEX/UNICAMP.

8 REFERENCES

[1] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

[2] F.A.M. Cappabianco, J.S. Ide, A.X. Falcão, and C.-S.R. Li. Automatic subcortical tissue segmentation of mr images using optimum-path forest clustering. In *International Conference on Image Processing (ICIP)*, pages 2653–2656, 2011.

[3] Yizong Cheng. Mean shift, mode seeking, and clustering. *TPAMI*, 17(8):790–799, 1995.

[4] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *JAIR*, 4:129–145, 1996.

[5] A. T. da Silva, A. X. Falcão, and L. P. Magalhães. A new CBIR approach based on relevance feedback and optimum-path forest classification. *Journal of WSCG*, pages 73–80, 2010.

[6] A. T. da Silva, A. X. Falcão, and L. P. Magalhães. Active learning paradigms for CBIR systems based on optimum-path forest classification. *Pattern Recognition*, 44:2971–2978, 2011.

[7] D. T. Davis and J. N. Hwang. Attentional focus training by boundary region data selection. In *Intern. Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 676–681, 1992.

[8] Faces. Biometrics Database Distribution. The Computer Vision Laboratory, University of Notre Dame, 2011. `www.nd.edu/~cvrl/CVRL/Data_Sets.html`.

[9] A. Holub, P. Perona, and M.C. Burl. Entropy-based active learning for object recognition. In *CVPRW*, pages 1–8, 2008.

[10] P. Jain and A. Kapoor. Active learning for large multi-class problems. In *IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, pages 762–769, 2009.

[11] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell. Gaussian Processes for Object Categorization. *International Journal of Computer Vision (IJCV)*, 88:169–188, 2010.

[12] X. Li, L. Wang, and E. Sung. Multi-label SVM Active Learning for Image Classification. In *International Conference on Image Processing (ICIP)*, volume 4, pages 2207–2210, 2004.

[13] J. P. Papa, A. X. Falcão, V. H.C. de Albuquerque, and J. M.R.S. Tavares. Efficient supervised optimum-path forest classification for large datasets. *Pattern Recognition*, 45:512–520, 2012.

[14] J. P. Papa, A. X. Falcão, and C. T. N. Suzuki. Supervised pattern classification based on optimum-path forest. *Intern. Journal of Imaging Systems and Technology (IJIST)*, 19(2):120–131, 2009.

[15] Pendigits. Pen-Based Recognition of Handwritten Digits Dataset. UCI - Machine Learning Repository, 2011. `archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits`.

[16] G.-J. Qi, X.-S. Hua, Y. Rui, J. Tang, and H.-J. Zhang. Two-dimensional multilabel active learning with an efficient online adaptation model for image classification. *IEEE Transact. on Pattern Analysis and Machine Intel.*, 31(10):1880–1897, 2009.

[17] L. M. Rocha, F. A. M. Cappabianco, and A. X. Falcão. Data clustering as an optimum-path forest problem with applications in image analysis. *Intern. Journal of Imaging Systems and Technology (IJIST)*, 19(2):50–68, 2009.

[18] S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *ICM*, pages 107–118. ACM, 2001.

[19] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research (JMLR)*, 2:45–66, 2002.

[20] R. Yan, J. Yang, and A. Hauptmann. Automatically labeling video data using multi-class active learning. In *IEEE Intern. Conference on Computer Vision (ICCV)*, volume 1, pages 516–523, 2003.

Exposing Proprietary Virtual Reality Software to Nontraditional Displays

Maik Mory
Otto-von-Guericke-University
maik.mory@ovgu.de

Steffen Masik
Fraunhofer IFF
steffen.masik@iff.fraunhofer.de

Richard Müller
University of Leipzig
rmueller@wifa.uni-leipzig.de

Veit Köppen
Otto-von-Guericke-University
veit.koeppen@ovgu.de

Abstract

Nontraditional displays just started their triumph. In contrast to traditional displays, which are plane and rectangular, they do not only differ in design and architecture; they also implicate different semantics and pragmatics in the rendering pipeline. We strive for a generic solution that couples legacy applications with nontraditional displays. In this paper, we present an architecture and a respective experiment, which exposes a proprietary virtual reality software to a 360 degree virtual environment. Therefore we introduce a rigorous master-slave design. The proposed architecture requires discussion of the following details: how to access a proprietary application's OpenGL stream; how to transmit the OpenGL stream efficiently in a clustered rendering setup; how to process the OpenGL stream for adaption to nontraditional display semantics; and how to deal with the arising code complexity, withal. Our design decisions are highly interdependent. The presented architecture overcomes limitations, which were implied by client-server design in earlier work. The proposed rigorous master-slave design is totally transparent to the client software, and reduces interdependencies between rendering software and rendering clusters. Thus, it inherently reduces network round trips and promotes the use of scalable multicast. Our architecture is tested in a reproducible experiment, which provides a qualitative proof of concept.

Keywords: Nontraditional Display, OpenGL, Distributed Rendering, Multicast, Interoperability, Generative Programming

1 INTRODUCTION

Nontraditional displays (e.g., CAVEs, powerwalls, domes) just started their triumph. In the 1990s, nontraditional displays were driven by special monolithic rendering hardware. Together with the advent of cheap general and graphics computation power driven by the computer games industry, research shifted towards rendering clusters made from off-the-shelf hardware, during the first decade of our century.

In contrast to traditional displays, which are plane and rectangular, nontraditional displays do not only differ in design and architecture; they implicate different semantics and pragmatics in the rendering pipeline, too. This paper discusses several solutions that were implemented to bring legacy software to nontraditional displays. Our predecessors inherited client-server semantics from the OpenGL specification. We present an implementation that rigorously uses master-slave semantics to enhance scalability of distributed rendering architectures, beside other minor optimizations. Throughout our discussion, we use a practical application scenario that is presented in this section's remainder together with a problem statement. Section 2 provides background on interoperability of distributed virtual reality software. An architecture, which enables node-based OpenGL stream processing with little interference to the communication between a central OpenGL client software and its local OpenGL server hardware, is presented in Sections 3 and 4. That our proposed architecture is feasible is proven with a reproducible experiment in Sections 5 and 6. Finally, we conclude with an outlook on future work.

1.1 Application Scenario

Today, every engineering process is supported by software. Any reasonable engineering software has a visualization component. Note, the visualization component is not necessarily a dominant or permanent element of the user interface; we only require it to be available. We choose Bitmanagement's BSContact [8] as an exemplary sample of generic information and geometry visualization. Basically, BSContact is a generic 3D file viewer for VRML and X3D. Most engineering software is able to export these data formats. Although, for us BSContact's primary functionality is not that relevant. For us it is important, that BSContact uses the most widespread patterns of three dimensional data visualization; it is proprietary; and it renders interactive geometry.

Several nontraditional displays exist in various dimensions and shapes. Most of them have an entertainment background, but some of them are also used for research and industrial applications. One of those systems is the ElbeDom located at the Fraunhofer Institute for Factory Operation and Automation (IFF) in Magdeburg, Germany. The ElbeDom is a large cylindrical virtual environment. It has been designed to satisfy the demand

for immersive virtual reality (VR) and massive multi-user collaboration in the areas of virtual manufacturing and factory planning. A detailed description of the system and a comparison of similar projection systems are provided in [22]. Basically, the ElbeDom is representative hardware for distributed, tiled rendering on curved screens.

Briefly, the ElbeDom's cylindrical screen is 6.5 m high and 16 m in diameter. The 330 m^2 screen is covered by six LDT G2 laser projectors. The projectors operate at 1600×1200 pixels. Thus, the viewer is surrounded by approximately eleven megapixels from -20° below horizon to +30° above horizon and full 360° in horizontal. Six so-called warping engines geometrically adjust and blend the six projector's pictures. Their input is generated by a cluster of six commodity nodes. Throughout this paper we call the six nodes slaves. Among other control and automation nodes, there is a dedicated headed node for the operator. We will call it the master in this paper. The commodity cluster is interconnected via off-the-shelf GBit Ethernet.

1.2 Problem Statement

Our primary goal is to provide interoperability between arbitrary proprietary virtual reality software and arbitrary nontraditional displays. Looking at this paper's application scenario alone, several questions arise. Bitmanagement's BSContact has been developed for traditional displays connected to local graphics hardware. Thus, one has to consider aspects of syntactical, semantical and pragmatical interoperability between the visualization software and the nontraditional display's rendering pipeline. Challenges, how to interface a proprietary software's visualization component and how to handle variant implementations of several hundred OpenGL functions frame our considerations.

2 BACKGROUND

Engineers describe system architectures in terms of components and interfaces. Depending on the engineer's domain and preferred method, what we call a component may be called an object, device, service, module, or other too. We focus on the domain of computer science. Therefore, a component is a definable software artifact. Connections of components are differentiated between tight couplings and loose couplings. Tight coupling exploits interdependencies and relations between components; the connected components are not supposed to be exchanged. A loose coupling minimizes dependencies and relations between the components to a well-defined specification of the interface; loosely coupled components tend to be exchangeable. In real life, couplings are not clearly the one or the other. Rather, real couplings distribute in a continuum with ideal loose coupling on one end and with ideal tight coupling on the other end. The distinction is made, whether an instance is more the one or the other.

Coupling components is the subject of interoperability. Interoperability is a field of active research. The most exhaustive, recent survey we know of was done by Manso et al. [17]. They declare seven levels of interoperability: technical, syntactic, semantic, pragmatic, dynamic, conceptual, and organizational. In our context it is sufficient to stick with a three level hierarchy of interoperability [14], which we briefly introduce as follows:

Syntactic interoperability is data exchange with a common set of symbols to which a formal grammar applies.

Semantic interoperability is information exchange with a shared, common vocabulary for interpretation of the syntactic terms.

Pragmatic interoperability is contextual exploitation of applications and services through shared knowledge.

Another aspect about couplings are messaging patterns. Most procedural, object-oriented, and distributed systems follow the client-server pattern. A server component provides an interface. A client component requires an interface. If a client's required interface and a server's provided interface are compatible, they can be connected. Then, the client uses the server. Servers or services can be stateful or stateless. If the server is stateless, the effect to a request depends on the request only. If the server is stateful, the reply depends on the request and on the server's state.

The Gang of Four [7] identified an extreme variant, where the request declares the client's interest in a series of replies – the observer pattern. Other names are one-way messaging, publish-subscribe, producer-consumer, or master-slave as we call it in this paper. Master-slave tends to be loosely coupled, because in an ideal implementation the consumer requires no knowledge about identity or number of the producers and vice versa as well, for example.

An interface definition covers a slice of the interoperability hierarchy. Most interface definitions in computer science, especially application programming interfaces' documentations focus on syntactic and semantic interoperability. Software developers usually delegate technical interoperability to electrical engineers, who design computer chips and network links. The upper half of the interoperability hierarchy usually is in the responsibility of software project's stakeholders.

In a system of n components, one may implement $O(n^2)$ adapters for each coupled pair of components. When there is a common concept, which is shared among several interfaces, established protocols and other

interface specifications are reused for multiple components. This we call an interoperability platform. In a system of n components, one implements $O(n)$ adapters between each component and the interoperability platform.

There has been vast work to establish interoperability platforms for VR applications. For example, Schumann [23] uses the high-level architecture (HLA) for syntactical interoperability among distributed simulations; Ošlejšek [21] tries to establish semantic interoperability with a unified scene graph definition. The crucial point in the design of an interoperability platform is the common concept shared between participants. In our observation, there are two types of interoperability platforms: those which declare and impose an artificial common concept; and those which find and exploit an existing common concept. We believe that the latter have better chances to succeed in software evolution. Looking at the abundance of VR software, there is one thing obviously common: OpenGL. The OpenGL specification defines syntax through function signatures together with a finite state machine and it defines semantics through human readable documentation for modules and functions.

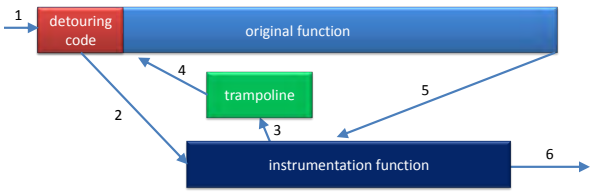
We are not the first ones who exploit the well supported OpenGL industry standard for interoperability. The commercial software products TechViz XL [2] and ICIDO's Capture [1] impressively show the potential. However, because they are closed-source they give little value to our discussion. During our discussion we mostly refer to selected aspects of Chromium [11, 16], Lumino [25], and BroadcastGL [13].

3 EXPOSING A PROPRIETARY APPLICATION'S OPENGL STREAM

In [18], the authors evaluate four techniques, how to intercept a proprietary application's invocations of the OpenGL API. Three out of the four techniques have been used in multi-hosted rendering before. The relink library technique (e.g., MPIglut [15], although it is not a node-based stream processor) cannot be applied to proprietary software. The replace dynamic library technique (e.g., Chromium [11]) is unreliable within MS Windows' dynamic library facility. The virtual device driver technique (e.g., VirtualBox [26]) does not scale for complex application scenarios. We prefer the binary interception technique because it is flexible and robust at once.

The binary interception technique was introduced by Hunt and Brubacher [12] to instrument and extend proprietary software. An injected intermediary manipulates the proprietary software's binary image at runtime. We illustrate the principle in Figure 1. For each function that should be instrumented, the intermediary installs what is referred to as detour. The installation procedure for a

(a) In unmodified software, the client requests the server function (1); then the server function returns a reply (2).



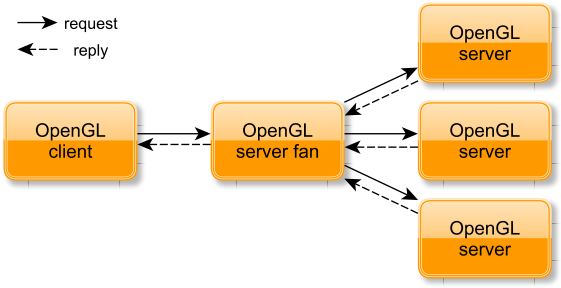
(b) In intercepted software, the detouring code redirects the instruction pointer to an instrumentation function (2). We internally invoke the original function through the trampoline (3,4,5). After processing the request-reply tuple, we return to the client (6).

Figure 1: Binary Interception

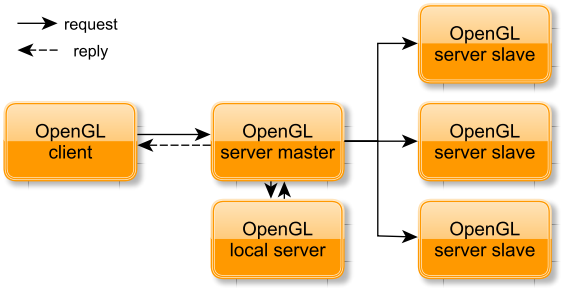
function overwrites the first bytes of the server's function with machine code, which detours the execution path to the intermediary's function. When the OpenGL client invokes an intercepted OpenGL server's function, the overlaid detouring code is executed instead of the original code. In effect, the client invokes the intermediary's function. The installation procedure produces a so-called trampoline function, which keeps the original server's function available. The trampoline contains a backup of the server function's machine code that was overwritten during installation of the detour and additional machine code that repatriates the execution path to the unmodified remainder of the server function's machine code. In effect, invocations of the trampoline delegate calls to the server.

Microsoft Windows' implementation of OpenGL, which is determined to be compatible with OpenGL version 1.1, provides 2400 functions (c.f., Section 6), which divide into 357 core functions, 1671 extension functions, and 372 alias functions. Core functions are provided in the `opengl32.dll`'s symbol table. We install interceptions for every core function during startup time before the application is able to access them. Extension functions are provided on the client's demand on the server through the `wglGetProcAddress` function. We install interceptions for every extension function when it is passed through the `wglGetProcAddress` function for the first time. What is known as alias functions are alias names for core or extension functions. In the data structure that tracks installed interceptions, alias functions are associated with their respective original functions.

In result, the complete OpenGL API is instrumented. Our instrumentation functions first transparently delegate the client's request to the original function through the trampoline. After the original function returned (item 5 in Figure 1b), the request-reply tuple is available to the instrumentation function from a totally transparent observation. In our further discussion, a published sequence of request-reply tuples we call the OpenGL



(a) Traditionally, the client's requests are fanned to the distributed renderer and replies are merged after a full round trip. The display's semantics and pragmatics are opaque to the client.



(b) With our approach, the server master uses a local server to gather replies with minimal latency. Then it publishes the requests with attached replies to the slaves without network round trips. The slaves adapt semantics and pragmatics transparently.

Figure 2: A node-based OpenGL stream processor that multicasts request-reply tuples has looser couplings.

stream. After publication of the new OpenGL stream element, we return to the original OpenGL client using the reply from the original server function. Thus, we have extracted the OpenGL stream from the client-server coupling with minimal interference.

4 FULL MULTICAST SEMANTICS FOR OPENGL STREAM DISTRIBUTION

Commonly, implementations of node-based, distributed OpenGL processing use unicast via TCP/IP for data distribution. We guess that this design decision has two major reasons. Back in the days of the first hype about rendering on commodity clusters (for a survey see Chen et al. [5]), TCP/IP was available, tried-and-tested, and well supported. Further, for those systems it is a basic assumption, that the invocation actually happens on the remote site; and thus, return values and argument alterations have to be propagated back from the distributed OpenGL server to the central client. Figure 2a illustrates this in terms of distributed systems. The OpenGL client's requests are fanned to multiple OpenGL servers. For operations with output, the server fan adapts the client's request for the remote servers and merges the remote servers' replies to a singular reply for the client. Thus, the nontraditional display's semantics and pragmatics are opaque to the client. As an example for opaque semantics, all of our predecessors tried to map the semantics of windowing and camera setup (e.g., the `glViewport` function) from their tiled display setups to the clients' calls.

In 2005, Ilmonen et al. [13] unveiled the potential of non-unicast stream distribution in the context of multi-tile rendering. They discovered, that unicast stream distribution does not scale with the number of tiles, because shared commands that are used by n tiles have to be sent n times. The more tiles a distributed graphics application uses, the more neighboring and blended regions share geometry data. Global state changes are shared between all tiles. Ilmonen et al. describe an experiment, where they use broadcast via UDP/IP for stream distribution and a TCP/IP backchannel to add reliability and congestion control. Their main contribution is an empirical proof, that broad- and multicast OpenGL stream distribution scales with the number of tiles in a centralized application with distributed graphics.

Lorenz et al. [16] implement a modification of Chromium, which uses multicast for parallelizing commands and unicast for serializing commands. In their reasoning, serializing commands are those commands that are unique to each remote server. Serializing commands are different with respect to the distinct peers, because they are adapted on the client's side of the network in the server fan. We argue that they could be parallel calls – and thus be appropriate for multicast distribution – if the adaption stage would be shifted from the server fan to the remote peers.

Ilmonen et al. [13] already shift adaption of the requests to the remote peers. In their discussion, they point out that commands which require merged replies from the distributed server stall the streaming. Neal et al. [20], who advance efficiency in multicast OpenGL stream distribution by applying compression techniques to the distributed stream, observe the same problem. They identify, that commands which have replies effectively are network round trips and hence cause blocking at the client. This leads us to the question: Is it really necessary to aggregate state from the distributed server? Neal et al. as well as Ilmonen et al. still use client-server semantics as it is defined in the OpenGL API specification. As an aside, they borrow a potential solution from prior work, that round trips may be avoided by state management [4].

Chromium [11] and Lumino [25], for example, implement state management. State management emulates the OpenGL state machine as a component of the stream processing framework. Stavrakakis et al. [25] claim that state management is necessary for two reasons: late joiners should be able to retrieve OpenGL machine's state; and operation accumulation can be used for compression of transferred data (i.e., a-priori-aggregated state of

the distributed server). However, state management is expensive. With emulation software, it is tedious to keep track with the original implementations in functionality and in performance. When Chromium introduced state management, they assumed that the client may run on a platform without graphics acceleration. Today, every host has basic graphics hardware acceleration or at least a good software implementation. Hence, we consider the topic of emulated state management obsolete. Even more, we explicitly recommend using the central application's local OpenGL implementation.

This yields a novel architecture for centralized applications with distributed graphics, which is depicted in Figure 2b. Our implementation of the intercepted OpenGL API, which we name Vanadium[1], first invokes the original local OpenGL server with unmodified commands from the client and returns unmodified output to the client. This renders Vanadium transparent in functional behavior to the OpenGL client and in appearance to the user at the host with the central application, as well. After the trampoline function has returned and before the decorator function returns (cf., Figure 1), the decorator publishes the OpenGL stream. In contrast to earlier work, the stream does not only contain opcode and relevant input arguments (i.e., the request), but also every output, like return values and referenced arrays (i.e., the reply). After publishing the stream, any further stream processing is asynchronous. Thus, there are no round trips on the network anymore.

We agree to Stavrakakis et al. [25], that there should be a possibility to join lately to the stream. Nevertheless, late joining is a very infrequent event. Thus, we abnegate the implementation cost and runtime cost of dedicated state management. In the infrequent case of a late join, we are able to retrieve the OpenGL machine state directly from the original driver vendor's implementation through the `glGet` function and other inspection functions; at least unless the client uses no deprecated technique like display lists. The joined slave uses its adaption (see Section 5) to map the master's late state to a consistent slave's state. Then, the stream processing can continue. Regarding stream compression, which is a topic to all distributed OpenGL renderers, because the network bandwidth bottleneck is very dominant, we refer to Neal's work [20] and Lorenz' work [16].

5 EXAMPLES FOR DETACHED PROCESSING OF THE OPENGL STREAM

In this section, we clarify the architecture shift that we propose in Section 4. Therefore, we describe the processing chain as we implemented it with Bitmanagement's BSContact as source and VDTC's ElbeDom as sink (cf., Section 1.1 and Figure 3).

Figure 3: VDTC's ElbeDom driven by Vanadium. The transparently distributed application is Bitmanagement's BSContact with a software visualization scene [19].

Please note, that we intercept the whole OpenGL and WGL API. In the decorator functions we first delegate the call to the original function via the trampoline. After the trampoline function returned with the output from the original server, the processing described in this section takes place. After our processing, the decorator function returns the values from the local original server invocation to the original client (cf., Figures 1 and 2).

First of all, we need to handle recursion. As we use binary interception, we get each and every call of the OpenGL API – literally. So, there are the calls actually made by the client; and there are recursive calls by the original server implementation to itself. For example, the `wglDescribePixelFormat` function calls itself; many functions call the `glFlush` function internally. We distinguish client calls and recursive server calls by tracking stack depth with a counter. When there is no active call from the client, the counter rests at minus one. During client's requests the counter is zero. When the server calls itself, the counter is greater than zero. Recursive calls are skipped in stream processing, because they reflect internal behavior of the original local server and thus do not matter. Now, the stream contains every invocation actually made by the client.

[1] Vanadium as used in the presented experiment is provided in the additional material. It will be open-sourced, soon.

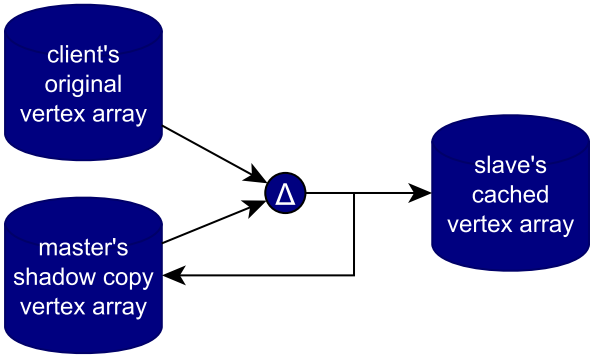


Figure 4: Vertex Array Cache – The master maintains a shadow copy of the client's vertex arrays. The vertex array transmission to the slave is differential.

Looking at the stream of client calls, there is a huge amount of calls that are irrelevant to the remote display. Most commercial OpenGL software use an off-screen rendering technique for auxiliary calculations, like mouse pointer ray collision testing or occlusion culling. In Bitmanagement's BSContact, a temporary viewport is used, that is overdrawn by visible content before the next SwapBuffers invocation. BSContact's hidden viewport is easily determinable, because it is always square (e.g., 100×100 pixels) and smaller than the window's rendering area. We skip all calls that are made to the finally invisible viewport. Then BSContact sets the viewport to the whole visible area and renders the visible content. We passthrough these calls for handling on the nontraditional screen. As can be seen from the invocation log, another viewport is set, that is always 86 pixels in height. Because we could not imagine a mapping from its two-dimensional content to the 360 degree screen of the ElbeDom, we skip the two-dimensional content, too. The considerations in this paragraph are highly interdependent with the client software and thus, have to be reconsidered for every new client.

To reduce bandwidth usage further, we apply a caching technique to the vertex array facility (Figure 4). To achieve this, our decorated `glDrawElements` function and related vertex array draw functions determine the array in CPU's RAM that should be drawn by pointer and by size. The master keeps a shadow copy that resembles the arrays in the slave's cache. We introduce cache management commands in the stream to advice the slave for modifications of its cache. Initially the master's shadow copy and the slave's cache are empty. When there is a new array (i.e., pointer is not in shadow copy mapping), it is added to the shadow copy and transmitted to the slaves. When the client draws a known array (i.e., pointer exists in shadow copy mapping) whose content is unchanged (i.e., client's array equals shadow copy array), the slave is told to use the cached array. When the client draws a known array whose content has changed (i.e., pointer exists in shadow copy mapping and client's array is not equal to shadow copy array), the changed array is transmitted to the slave before usage. At the slave, the modified array replaces the respective array, because obviously the client discarded the old content before. When distributing the arrays and when referring to them in draw commands, the master uses handles that are derived from its shadow copy index. During cache management commands, the slave maintains a mapping between the master's handles and the cache's pointers. After cache synchronization the master emits the draw command. Then the slave uses its master's-handle-to-slave's-pointer mapping to invoke the draw command with valid data. This simple caching technique doubles the client application's memory consumption with respect to its vertex arrays. We do not recommend the use of hashes, because collisions in the index may corrupt the stream fatally [16]. For us, usage of the `memcmp` function worked out by reducing network bandwidth at negligibly increased CPU load.

For networking, we use ØMQ [10], which provides us with superior inter-thread, inter-process, cluster-wide, and world-wide messaging. One command is one message. For each command in OpenGL's API specification, we derive a struct, which contains the opcode, any value arguments, and the return value if applicable. Array arguments are packed into submessages. Because ZeroMQ takes care of message sizes robustly, therewith we significantly reduce any risk associated with wrong array sizes in C/C++. As an optimization, we exploit that call-by-value arguments already are packed in the stack. Thus, we only need to copy a slice from the stack into the corresponding slice of the message buffer struct. ZeroMQ offers various reliable multicast protocols for data distribution. After message transmission through ZeroMQ to the slaves, the commands are dispatched to handler functions based on their opcode. The slave's default handler implementation directly mimics the client node's invocation. Some handler functions are modified to implement adaption of the stream at slave side.

Comparable to the mapping of array pointers the slave implements a mapping of OpenGL names. In OpenGL's terminology, names are numeric identifiers for objects in the OpenGL state machine. For example, the `glGenTextures` function outputs integers to the client, which the client should use to identify and refer textures unambiguously. Usually, equally replayed commands should yield equal names. However, we cannot guarantee that for heterogeneous environments, for late joiners, and when splicing command streams. The mapping mechanism is simple. There are functions that generate (i.e., output) names and there are functions that use (i.e., input) names. The slave maintains an associative array with the master's names as keys and the slave's names as values. Please remember, that the master includes invocations' outputs in the stream. When

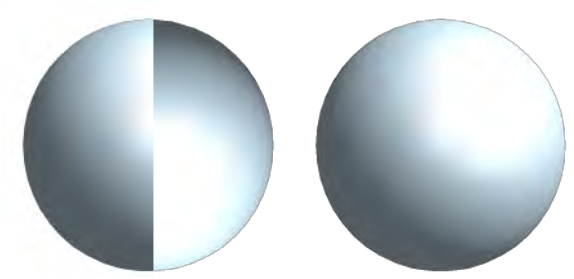


Figure 5: If lighting is calculated relative to the camera and the camera has different poses on different tiles then the light's pose is different on each tile (left). Reposing light sources with respect to the tiles' camera pose differences compensates the visual inconsistency (right).

a command that generates names is called, the slave extracts the master's names from the arrived message. It replays the command, which yields the slave's names. Then the pair is added to the map. When a name is used, the message refers the master's name. The slave decodes the master's name to its local name using the map. We call this mechanism name mapping.

The ElbeDom is an exemplary virtual environment. The cylindrical, surrounding screen has little potential for two-dimensional WIMP semantics. Hence, we do not even try to map the client's WIMP semantics to the system's VE semantics. The filtering stage at the master yields a singular stream with visually relevant content only. Therefore, calls to the `glViewport` function, which refer to the whole visible window area at the master, are mapped to fullscreen rendering at the slaves. This leaves us to adapt the camera pose. The appropriate place in the stream to achieve camera adaption is client-software-specific. With Bitmanagement's BSContact, the most robust solution is to modify invocations of the `glLoadIdentity` function and the `glLoadMatrix` function where the model view matrix is selected. There we premultiply the tile's camera orientation. This gives us the basic adaption to the ElbeDom's 360 degree view. Additionally, the ElbeDom's warping and blending facility specifies asymmetric frusta for each tile. At the `glFrustum` function we discard the client's inputs for a symmetric view into the window and overwrite with the tile's asymmetric frustum configuration. This completes camera adaption to the ElbeDom's interleaved frustum configuration.

The adaption pipeline is completed by a pragmatic adaption of lighting. Like most other OpenGL software, BSContact uses lighting to give three-dimensional impression. BSContact's lighting is designed relative to the camera. Because we rotate the world to adapt the camera position, the lighting is inconsistent between the tiles (cf., Figure 5). Hence, we install a filter on the `glLight` function family, which applies the tile's camera pose to the position and direction lighting parameters. Thus, the lighting is consistent through all of the ElbeDom's six tiles.

6 HANDLING CODE COMPLEXITY

The functionality that we describe in this paper handles several hundred functions from the OpenGL API. Moreover, we talk about variant functionality. During analysis of the application's OpenGL stream, we need a variant that logs the OpenGL stream to disk. The sender is a variant that implements serialization. The receivers have to implement deserialization. Processing of the OpenGL streams yields building blocks (i.e., filters, adapters), that ideally should be individually and independently reusable with a broad variety of virtual reality software and virtual environments. Nevertheless, we expect that processing nodes need to be tailored with respect to functional and non-functional behavior (cf., Siegmund et. al. [24]) as soon as more than one application will be supported. At a first glance, this requires a codebase of several thousand repetitive functions. At the second glance, we see two core problems: repetitiveness and variability.

Under the bottom line, we deal with a component oriented system, where the components stem from a software system family. Such a family includes a number of systems that are similar enough from an architectural point of view to be assembled from a common set of components. We achieve development efficiency through Generative Programming (GP) [6]. The main goal of GP is to generate a partial or an entire software system automatically from such implementation components. The requirements for the desired result, i.e. the generate, are defined in a domain specific language (DSL). A DSL is a specialized and problem-oriented language for domain experts to specify concrete members of a system family. This specification is processed by a generator, which automatically builds the system by combining components according to configuration knowledge. The Generative Domain Model (GDM) in Fig. 6 illustrates the concept of this paradigm establishing a relationship between the basic terms of GP. It consists of the problem space, the solution space, and the configuration knowledge mapping both spaces. The problem space includes domain specific concepts and features to specify requirements by means of one or more DSL(s). The solution space offers elementary and reusable implementation components corresponding to the system family architecture. The configuration knowledge comprises illegal feature combinations, default settings, construction rules, and optimizations as well as related information. In order to instantiate this theoretical concept, we perform a technology projection. Therefore, we identify concrete techniques for the elements of the GDM.

In particular, we enhanced Microsoft Visual C++ at Visual Studio 2010's prebuild stage. First, our Python

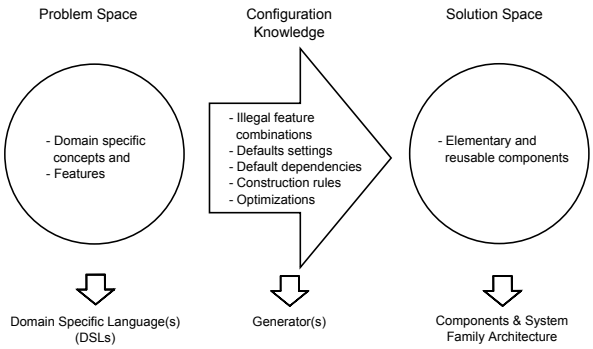


Figure 6: Generative Domain Model [6]

script parses the OpenGL API's formal specification. At the time of our experiment, the OpenGL specification (revision 12819) defines 2269 functions. Microsoft Windows' OpenGL windowing system (WGL) (revision 10796) adds 131 function definitions. Thus, an OpenGL application on Windows has access to a repository of 2400 functions. Secondly, within our Python-based domain specific language, a feature configuration is modeled. Both models together are applied to a template engine, which generates C++ source code. Then, the prebuild step, or code generation step respectively, terminates and the Visual C++ tool chain builds the executable software artifacts. Applying the means of the generative paradigm, repetitive development tasks are automated and the high variability of the virtual reality domain is made manageable.

7 CONCLUSION AND FUTURE WORK

We presented an architecture for node-based OpenGL stream processing. It is highly interoperable with proprietary and legacy software, because we use a robust technique for function interception, and especially because we adapt from OpenGL's client-server pattern to a master-slave pattern, which is more feasible for stream processing. The adaption is as transparent as possible to the client. Thereby, the proposed architecture removes any interdependencies between rendering software and rendering clusters. We show that there is no source code access to the application required. The reduced interdependency promotes to use scalable multicast, and removes network roundtrips. The architecture is tested by a reproducible experiment, which we comprehensively described in Section 5.

For sake of clarity, we focused on the combination of one virtual reality software with one nontraditional display. We want to generalize our approach of course. At the time of this writing, we are experimenting with other rendering software, and we are experimenting with other nontraditional displays, which have different pragmatics than the ElbeDom. One open question we are researching is how to deal with frustum culling and occlusion culling in legacy software. These techniques are highly optimized towards traditional displays. With BSContact, we could switch them off through its ActiveX interface. If culling was not disengageable, further investigation would be necessary. In the long term we are curious, if fully programmable pipelines may yield new rendering pipeline semantics, will our approach scale?

With our proposed rigorous master-slave design, the nodes of an OpenGL stream processor are coupled more loosely than in competing frameworks. Therefore, our architecture supports development and recombination of OpenGL stream processor nodes. For example, we would be glad to see an effect processing engine (e.g., Haringer and Beckaus [9], or Brennecke et al. [3]) applied to our OpenGL stream processing approach. In summary, we see great potential for innovative use cases in entertainment and engineering.

ACKNOWLEDGEMENTS

We are thankful to Fraunhofer IFF's Virtual Development and Training Center (VDTC), which provided its ElbeDom for our experiments.

REFERENCES

[1] IC:IDO – The Visual Decision Company. `http://www.icido.de/PDF/ICIDO_Broschuere_A4.pdf`, Stuttgart, November 2006. In German.

[2] Emanuela Boutin-Boila. TechViz XL, March 2010.

[3] Angela Brennecke, Christian Panzer, and Stefan Schlechtweg. vSLRcam – Taking Pictures in Virtual Environments. In *WSCG (Journal Papers)*, pages 9–16, 2008.

[4] Ian Buck, Greg Humphreys, and Pat Hanrahan. Tracking graphics state for networked rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '00, pages 87–95, New York, NY, USA, 2000. ACM.

[5] Yuqun Chen, Han Chen, Douglas W. Clark, Zhiyan Liu, Grant Wallace, and Kai Li. Software Environments For Cluster-Based Display Systems. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, CCGRID '01, pages 202–210, Washington, DC, USA, 2001. IEEE Computer Society.

[6] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming Methods, Tools and Applications*. Addison-Wesley, 2000.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.

[8] Bitmanagement Software GmbH. BS Contact 8.0. `http://www.bitmanagement.de`.

[9] Matthias Haringer and Steffi Beckhaus. Dynamic Visual Effects for Virtual Environments. In *WSCG (Full Papers)*, pages 49–56, 2010.

[10] Pieter Hintjens. *ØMQ – The Guide*. iMatix Corporation, `http://zguide.zeromq.org/`.

[11] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia '08, pages 43:1–43:10, New York, NY, USA, 2008. ACM.

[12] Galen Hunt and Doug Brubacher. Detours: Binary Interception of Win32 Functions. In *Proceedings of the 3rd conference on USENIX Windows NT Symposium - Volume 3*, WINSYM '99, page 14, Berkeley, CA, USA, 1999. USENIX Association.

[13] Tommi Ilmonen, Markku Reunanen, and Petteri Kontio. Broadcast GL: An Alternative Method for Distributing OpenGL API Calls to Multiple Rendering Slaves. In *WSCG (Journal Papers)*, pages 65–72, 2005.

[14] Veit Köppen and Gunter Saake. Einsatz von Virtueller Realität im Prozessmanagement. *Industrie Management*, 2:49–53, 2010. In German.

[15] Orion Sky Lawlor, Matthew Page, and Jon Genetti. MPIglut: Powerwall Programming Made Easier. In *WSCG (Journal Papers)*, pages 137–144, 2008.

[16] Mario Lorenz, Guido Brunnett, and Marcel Heinz. Driving Tiled Displays with an Extended Chromium System Based on Stream Cached Multicast Communication. *Parallel Computing*, 33(6):438 – 466, 2007. Parallel Graphics and Visualization.

[17] Miguel-Ángel Manso, Monica Wachowicz, and Miguel-Ángel Bernabé. Towards an Integrated Model of Interoperability for Spatial Data Infrastructures. *Transactions in GIS*, 13(1):43–67, 2009.

[18] Maik Mory, Mario Pukall, Veit Köppen, and Gunter Saake. Evaluation of Techniques for the Instrumentation and Extension of Proprietary OpenGL Applications. In *2nd International ACM/GI Workshop on Digital Engineering (IWDE)*, pages 50–57, Magdeburg, Germany, 2011.

[19] Richard Müller, Pascal Kovacs, Jan Schilbach, and Ulrich Eisenecker. Generative Software Visualizaion: Automatic Generation of User-Specific Visualisations. In *2nd International ACM/GI Workshop on Digital Engineering (IWDE)*, pages 45–49, Magdeburg, Germany, 2011.

[20] Braden Neal, Paul Hunkin, and Antony McGregor. Distributed OpenGL Rendering in Network Bandwidth Constrained Environments. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *EGPGV*, pages 21–29. Eurographics Association, 2011.

[21] Radek Ošlejšek. Virtual Scene as a Software Component. In *WSCG (Posters)*, pages 33–36, 2008.

[22] Wolfram Schoor, Steffen Masik, Marc Hofmann, Rüdiger Mecke, and Gerhard Müller. ElbeDom: 360 Degree Full Immersive Laser Projection System. In *Virtual Environments 2007 - IPTEGVE 2007 - Short Papers and Posters*, pages 15–20, 2007.

[23] Marco Schumann. *Architektur und Applikation verteilter, VR-basierter Trainingssysteme*. Dissertation, Otto-von-Guericke-University Magdeburg, Magdeburg, November 2009. In German.

[24] Norbert Siegmund, Martin Kuhlemann, Sven Apel, and Mario Pukall. Optimizing Non-functional Properties of Software Product Lines by means of Refactorings. In *Proceedings of Workshop Variability Modelling of Software-intensive Systems (VaMoS)*, pages 115–122, 2010.

[25] John Stavrakakis, Masahiro Takatsuka, Zhen-Jock Lau, and Nick Lowe. Exposing Application Graphics to a Dynamic Heterogeneous Network. In *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, WSCG '2006, pages 71–78, 2006.

[26] VirtualBox Service Desk. #475 – 3D acceleration support for VBox guests. `http://www.virtualbox.org/ticket/475` Accessed 2011-10-10.

Image Abstraction with Cartoonlike Shade Representation

GyeongRok Lee* HoChang Lee* TaeMin Lee* KyungHyun Yoon**

Chung-Ang University
Department for Computer
Science and Engineering
Korea (156-756), Seoul

starz | fanpanic | kevinlee @ cglab.cau.ac.kr*

khyoon @ cau.ac.kr **

ABSTRACT

Luminance quantization, which maps the luminance values of an image to discrete levels, is widely used for image abstraction and the expression of a cartoonlike effect. Existing luminance quantization techniques use each pixel's luminance value separately, leading to a noisy image. Additionally, they do not take the shape of the imaged object into consideration. Thus, they suffer limitations in terms of cartoonlike shade representation.

We propose a new luminance quantization algorithm that takes into account the shape of the image. We extract the silhouette from the image, compute edge-distance values, and incorporate this information intothe process of luminance quantization. We integrate the luminance values of neighboring pixels using an anisotropic filter, using gradient information for this filtering. As a result, ourquantized image is superior to that given by existing techniques.

Keywords

Image Abstraction, Image cartoonize, cartoonlike shade generation, luminance quantization

1. INTRODUCTION

The purpose of image abstraction is to simplify color and shape. In particular, image cartooning makes images look like a cartoon. Abstracted imagesmake it easier to recognize a scene,because their color and shape information is very simple. For example cartoon is easier to recognize than real scene, so it is often used for children's contents. Because of this advantage, abstracted images are widely used in books and movies. Cartoon images have the following features:

1. Simple color and shade (not noisy)

2. Silhouette-like object shade

To express these features, we use luminance quantization, a simple and fast non-photorealistic rendering (NPR) technique. Conventional approaches change an image's luminance values to certain discretelevels to give simplified images. However, they use the luminance values of individual pixels,so they are limited in representing the shade that reflects the shape of the object, especially since real-world objects are influenced by complex lighting. So, conventional approaches can't represent neat shade like a cartoon.

We propose an improved luminance quantization algorithm and image cartooning process. Our algorithm takes into account the shape of the depicted object. We extract the silhouette of the object and compute an edge-distance map, and then compute the gradient values of the pixels to create an anisotropic filter. Then we integrate neighboring luminance valuesusing this filter. Our algorithm thus results in an image that is less noisy and has cartoon-like shade.

Our main contributions are the following:we represent cartoon-like shade by considering the shape information of the object, and we improve the quality of the luminance quantization image by integrating neighboring luminance values using anisotropic filtering.

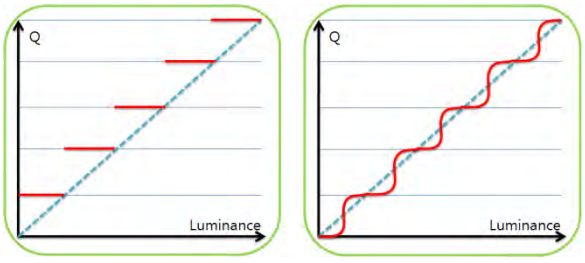
2. RELATEDWORK

There are several NPR approaches to abstract the color in a 2D image. The most conventional approach is image segmentation and express with average

color of region, which can simplifies image color but cannot preserve the original shape and detail.

More recently, filtering-based methods for image abstraction have gained popularity. The filtering approach is simple and very fast, so it is used widely. Filters include the blurring filter, the Kuwahara filter [Kuw76a], and [Pap07a] that is a generalization of various Kuwahara filters. Newer approaches include the bilateral filter [Tom98a] and the anisotropic Kuwahara filter. Thebilateral filter takes into account color and edge information, so it preserves the original shape. It is also very fast; Winnemoller et al. [Win06a] use this filter for video abstraction. The anisotropic Kuwahara filter expresses a painting effect by using an anisotropic filter within an existing Kuwahara filter. The above approaches express a cartoon-like effect through color simplification, but cannot express cartoon-like shad. For this, we use luminance quantization.

Quantization is a technique that is widely used with video, image, and audio. The original quantization [Cho89a, Zha95a] is a compression technique that is pursuing the save input information and reduce storage space. Apply this, Image quantization is used to reduce the cost on compressionand transmission costs by reducing the color information of the image.And quantization technique was used to representthe shading effect in NPR, which is luminance quantization. Conventional luminance quantization techniques individually change the luminance value of each pixel to a representative value. This leads to noise in the result image due to rapid stylization of luminance at the boundary (Fig.1-a). Therefore, in [Win06a] the tangent function is used to represent a smooth boundary and a natural transition between each step, solving the problem of sharp differences between the steps (Fig.1-b). This technique is simple and presentsthe shading effect well, so it has been widely used to express cartoon effects. However, the study in [Win06a] considers only the luminance information of individual pixels too , and thus does not reflect the morphological information of the image.



(a) **(b)**

Figure 1. (a) General luminance quantization (b) [Win06a]'s luminance quantization

Shape information is very important featurefor image abstraction. Many researchers have studied it: Kang et al. [Kan07a] proposed the edge tangent flow (ETF) technique that applies gradients to express a coherent line effect. Their method was extended to [Kan08a], which simplifies the shape and color of the image. Hertzmann [Her98a] and Hays et al. [Hay04a] determined the direction of the stroke by considering the form of the image to express the painting effect. Hochang et al. [Lee10a] utilized extra information on the image gradient from texture transfer and suggested a texture transfer technique with the direction. In their research on mosaics [Hau01a], Hausner used edge information to follow the direction ofthe edge to determine the direction of the tile. Although there have been many 2D-based studies that consider the shape of images, the form of the object has not been taken into account for the study of images' shade effects.

3. SHAPE-DEPENDENT LUMINANCEQUANTIZATION

Fig.2 shows our system flow. We use a 2D image as an input and apply Mean Curvature Flow(MCF) [Kan08a]. This gives an image with simplified color and shape. After that, we perform two steps. Firstly, we analyze the shape of the object. To do this, the silhouette is extracted and the distance of each pixel from the silhouette is calculated. In this step, flow information is extracted too.This step performed only once. After that, we integrate the luminance values of neighboring pixels using flow information, perform luminance quantization and modify luminance values using silhouette distance values. This step performed iteratively for all pixels of image. Finally, we apply MCF to shade shape clear up. Through these steps, we generate an image that expresses a cartoon-like shading effect.

3.1 Shape analysis

In order to express shade that reflectsshape of object, the shape information must be analyzed. Edge information is commonly used to reflect the shape of image. General edge detection techniques for image processing would extract too much information, so it is not suitable to extract result image which reflect shape like cartoons. Therefore in this study, we used the silhouette of the image and selected significant edge as shape information.

It is a simple matter to get the silhouette line of an object in a 3D environment since each object is saved in a separate space. Because of this, it is easy for extract boundary. In a 2D image, it is difficult to extract the silhouette of the foreground object since there is no depth information. (so, this is not perfect) We use mean-shift segmentation (result shown in Fig. 3-b) followed by user selection of the foreground

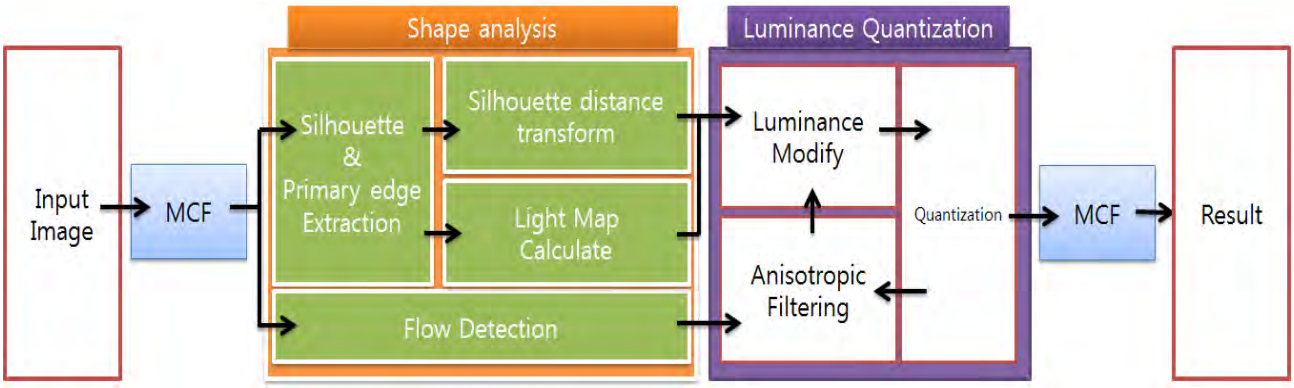
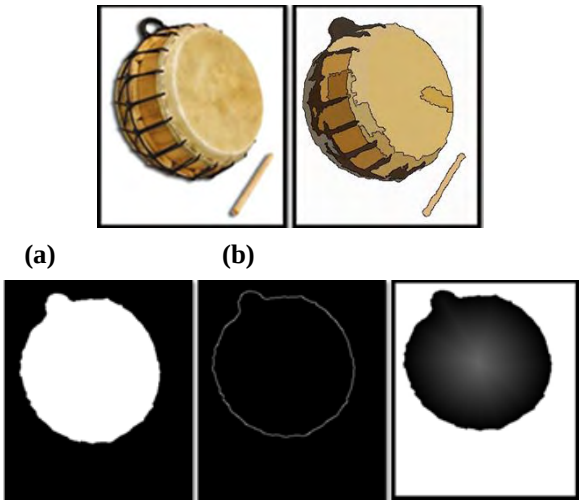


Figure 2. System Flow

object (Fig.3-c), which is then displayed as a white object on a black background

Performing an edge detection procedure gives the silhouette line (Fig.3-d). We then apply the edge-distance transform, an algorithm that extracts the distance between a pixel and the edge closest to it, on the pixels of the foreground object. This leads to an edge-distance map (Fig.3-e). The value of each pixel of the edge-distance map has been normalized to lie between 0 and 1.Thisedge-distance is used as an additional feature for luminance quantization, which will be covered later in this paper.



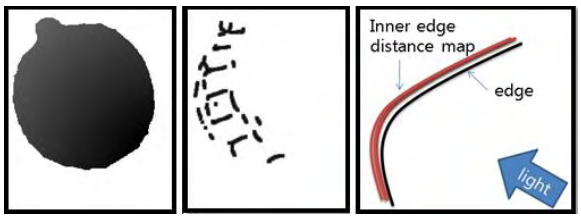
(a) **(b)**

(c) **(d)** **(e)**

Figure 3. (a)original image (b) segmented image (c) extracted object (d) silhouette line (e) distance map

In the cartoon, there are not only silhouette-like shades but also other shades, for example creases of cloth and important curves. To express these shades, we calculate the direction of light and extract the significant inner edge. we suppose that strong edge is significant edge. Strong edges have strong gradient value, so it can be remained after apply smoothing. Use this, we apply Gaussian smoothing to image firstly. after that, we extract edge and use this as a significant edge. Using a method used to calculate light direction in photography and videography, we use the highlight points (as seen in Fig.4)to calculate the direction of the light. This allows us to generate a virtual light source. After that, we extract the inner edge. Because the edge image is very complex when we use normal edge detection, we use a smoothed image as input of edge detection algorithm. Then we rule some edges which near to silhouette out inner edge.

Figure 4. light direction in the photography and videography



(a) **(b)** **(c)**

Figure 5. (a) light map (b) inner edge distance map (c) concept of inner edge distance

We can get two important features from the light point and inner edge. One is the light-distance map and the other is the inner edge-distance map. The light-distance map (Figure 5-a) is a map of the

normalized distance between each object pixel and the light point. We use the light map to prevent omni-directional shade generation, in other words, it enables us to generate shade on only one side of the object.

(a) (b) (c)

Figure 6. (a) edge image (b) normal edge distance map (c) example of edge movement

e calculatethe inner edge-distance map using a modified edge-distance transform. As we see in (Fig.6–a) and (Fig6-b), a normal distance transform gives distance values on both sides of an edge. Further, these values are spread over large area. This would lead to an unnatural shade. Hence, we modify the edge-distance transform to generate a narrow distance value, and we use a moved edge image made by moving the edge to the opposite side of the light (Fig.6–c). This gives the inner edge distance map. We see this in (Fig.5–b) and (Fig.5-c)

3.2 Luminance Modification and Quantization

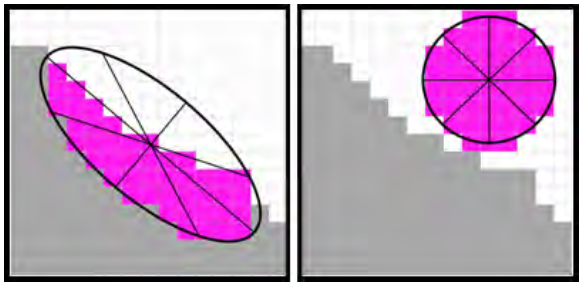
In this step, we apply two processes to all the pixels of image iteratively. The first is the use of an anisotropic filter to merge the luminance of neighboring pixels.This filter**i**ng applied anisotropic filter to pixel which have strong gradient. For find gradient, we use ETF field. On the contrast, if pixel do not have strong gradient, filter will be isotropic like (Fig.7) and (Fig.8)The second is the modification of luminance values using the silhouette distance map, the inner edge distance map and the light map. Finally, we apply quantization.

If we were to perform luminance quantization on the original input image, the result would be noisy because luminance values are not distributed evenly across an image. We need a coherent luminance distribution, for which we use a kernel that integrates the luminance values of neighboring pixels. We perform anisotropic filtering which takes into account image flow for the removal of image noise.

To facilitate image flow information, we use [Kan07a]'s method to utilize ETF (Fig.7-b). ETF, which interpolates gradient values, can extract more accurate and coherent pixel directions than other, more general methods that use Sobel or Laplacian filters. From the extracted ETF values, we calculate the structure tensor using x,y direction information. Using this matrix, we deform the circle filter to an anisotropic form. This filter reflects the flow of pixels. (Fig.7) is the ETF field obtained throughline integral convolution (LIC) [Cab93a], showing the orientation of each pixel in the image. In Fig.8 we can see how the anisotropic filter is applied. The luminance of the purple region is used as P in equation (1) below.

(a) (b)

Figure 7. (a) Original image (b)ETF field



(a) (b)

Figure 8. (a) Anisotropic filtering (b) Isotropic filtering

We use quantization to express the shading effect on the smoothened input image. In order to express a shade thatfollows the borderline, we define fallowing rule.

- Darken the luminance as the silhouette gets closer, lighten it as the silhouette gets further.

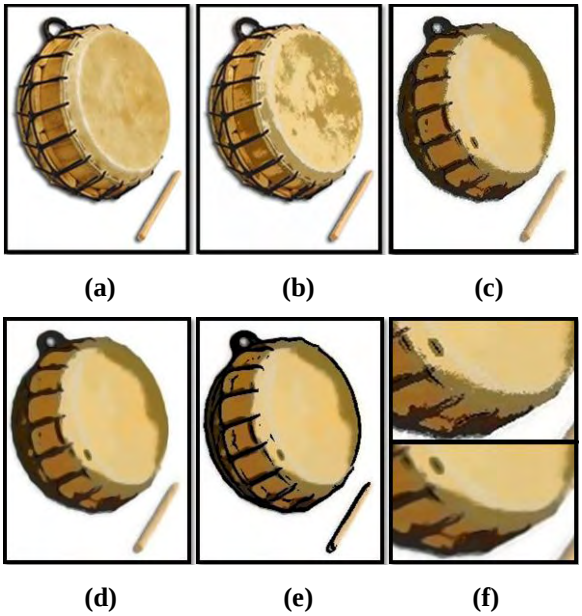
If the edge-distance value of the current pixel (P) is P_{dis} and its luminance value is P_{lum} we calculate $\tilde{P}_{lum}$ as follows:

$$\tilde{P}_{lum} = P_{lum} * (1 + W) \quad (1)$$

$$W = \log_{10}(k / W_{IL}) / 2 * LW \quad (2)$$

$$k = \frac{((P_{dis} * 100) + 1)}{((EF * 100) + 1)} \quad (3)$$

In order to calculate the luminance value that reflects the current luminance information and the information on the distance to the silhouette, we applied weighting based on the log function. Formula (1) shows the process of obtaining the weights W. We have used the effect control factor (EF) to adjust the value for log function. EF is the control variable that can adjust the distance value from the silhouette to where the shading effect is applied. The value of EF is between 0 and 1.As EF approaches 1, the value of K becomes somewhere between 0-1.Because P_{dis} is normalized into 0 - 1. Hence, the weighting value w is always negative and the final result is darker. Conversely, if the value of EF equals 0, the value of K is between 1 and100 and the weighting gets a positive value.0.25 is generally used as the value of EF. We divide this value k by W_{IL}, the inner edge-distance value. Through this we can generate shade on the inner edge region. As the value range of log(K) is -2 to 2, we use log(K) divided by 2. We multiple that value by light weightLW . We can get this value from the light map. The effect of this parameter can be seen in Fig.13.



(a) (b) (c)

(d) (e) (f)

Figure 9. Our results

(a)Originalimage (b)Conventional QT

(c)Our method (not MCF) (d)*(c)* + MCF

(e) *(d)*+Line (f) detail from *(c)* and *(d)*

Finally, the obtained $\tilde{P}_{lum}$ value is used as the input to the quantization process. We use the method used by Winnemoller et al. (Fig.1-(b)). Through this process, we get shade which is influenced by the shape. Finally, we apply MCF again for the arrangement of the shade.

4. RESULTS

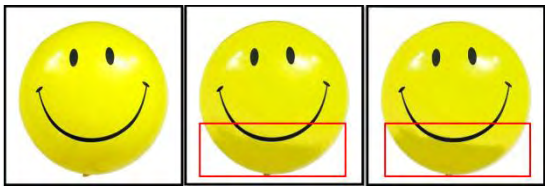
(Fig.9-a) is the input image, (Fig.9-b) is the result of conventional luminance quantization. (Fig.9-c) is the result of our anisotropic filtering and quantization. (Fig.9-c) maintains coherence among neighboring pixels and we can see the shade, it is scattered. (Fig.9-d) is the result of applying the MCF to (Fig.9-c). in this image we can see much better shade, although it is blurred. In (Fig.9-e), we enhance the edges. In the (Fig.9-f) we clearly see the shade on the object and we can check that shade arrangement by MCF.

(Fig.10) shows the result for different values of EF. A larger EF value results in an image with a larger shaded area.

(a) 0.05 (b) 0.25 (c) 0.75

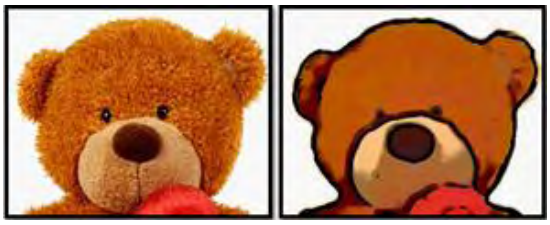
Figure 10. Results by EF value

We compare our results with conventional methods in (Fig.11).(Fig.11-a) is input image, (Fig.11-b) uses conventional luminance quantization. (Fig.11-c), which is the result of applying the quantization described by (Fig.1-b), shows superior definition in the shaded area compared to (Fig.11-b). (Fig.11-d), which is our result, better represents object shape than conventional approaches. (Fig.11-e) is the result of the procedure described in [Win06a]. (Fig.11-f) is the result of using the anisotropic Kuwahara filter. Compared with these, our result exhibits shade which reflects the object's shape. From (Fig.11-g) to (Fig.11-j) are other comparisons.



(a) Input (b) Figure1-a (c) Figure 1-b

(d) Our result (e) [Win06a] (f) [Kyp11a]



(g) Input (h) Our result

(i) [Win06a] (j) [Kyp11a]

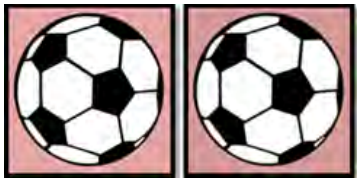
Figure 11. compare results with conventional researches

In (Fig.12), where the luminance is uniform, the shading is not expressed under conventional quantization. However, our method can generate shade.

(a) Input (b) Conventional (c) Our result

Figure 12. result of same-luminance image

(Fig.13) shows the results of varying the value of LW. (Fig.13-a) is input image. (Fig.13-b) is result of conventional luminance quantization. In (Fig.13-c), the light is at the top right; in (Fig.13-d), it is at the top left. In (Fig.13-e), the light is focused at the center of the object. Thus, we can control the shade effect through this parameter. We use this by making distance map. If user select light spot, system calculate light-distance map. This distance value is LW value. And if we don't use LW, we will get result



(a) (b)

(c) (d) (e)

Figure 13. Results by LW value

image like (Fig.13-e). That has omni-directional shade. And (Fig.15) is our additional results.

5. Conclusion and Future Work

In this paper, we proposed an image abstraction technique that is based onluminance quantization and takes into account the shape information of an image. We extracted the silhouette of the input image and utilized each pixel'sedge-distance from the silhouette as an additional feature. We were able to remove the noise while preserving the shape through anisotropic filtering. Through our algorithm, we could expressthe shading effects reflecting the shape of the image better than previous approaches.

The proposed algorithm has two advantages in terms ofthe shading effect compared to conventional luminance quantization. Firstly, the output we have producedmaintainsbetterconsistency among neighborhood pixels. This is because of anisotropic filtering, which considers the flow of the image in contrast to simple isotropic filtering. Thus, the noise was reduced compared to previous studies. The algorithm should be applicable to video content as well. Secondly, the shadow that tags along the shape of the image can be generated. Therefore, the shadow effect is more cartoonlike than in previous studies. In addition, through a simple parameter, the degree of shading can be adjusted and this can easily express variety of effects.

However, our algorithm has several limitations. Firstly, it is sensitive to the background of the image.

If the background is complicated, the result of the segmentation step is poor and object extraction becomes difficult.In (Fig14-a), we can see this limitation. To overcome this, we could use superior segmentation techniques (e.g. matting-based techniques). Secondly, our algorithm has problem when applied to very bright or dark image. Although we use light-map, we can't generate shade. (Fig14-b) Because dark image is very insensitive to luminance change.on the contrary, bright image is very sensitive to luminance change. So, it generates omni-directional shade. Thirdly, the execution time of our algorithm is very sensitive to the size of the anisotropic kernel. However, our algorithm is computationally parallelizable because it works on individual pixels;thus we can overcome this limitation by using a graphics processing unit (GPU). Finally, our algorithm needs some user interactions. If there are automatic object extract techniques, it can be improved

Figure 14. limitation

(a) left-original image, right-segmented image

(b) upper-light map, lower-result image

Acknowledgments

This work was supported by a Korean Science and Engineering Foundation(KOSEF) grant funded by the Korean government(MEST) (No.20110018616). And this work was also supported by a Seoul R&BD Program(PA110079M093171).

Figure 15. Additional results(arrow : light)

6. REFERENCES

[Bra97a] J.P Braquelaire, L.Brun : Comparison and Optimization of Methods of Color Image Quantization, IEEE transactions on image processing, Vol.6, No.7, pp.1048-1052, 1997.

[Cab93a] B. Cabral, L.C. Leedom :Imageing Vector Field using Line Integral Convolution, SIGGRAPH conference proceedings, pp.263-270, 1993.

[Cel09a] M.E. Celebi : Fast color quantization using weightedsort-means clustering. JOSA A, Vol. 26, No.11, pp.2434-244, 2009.

[Cha94a] N. Chaaklha, W.C. Tan, Teresa H.Y.Meng : Color Quantization of Images Based on Human Vision Perception. IEEE International Conference, 1994.

[Cho89a] P. CHOU, T. LOOKABAUGH, R.MGRAY : Entropy-Constrained Vector Quantization. IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, Vol. 31, No.1, pp.31-42, 1989.

[Hau01a] A.Hausner : Simulating Decorative Mosaics. SIGGRAPH conference proceedings, pp.573-580, 2001.

[Hay04a] J. Hays, I. Essa: Image and Video Based Painterly Animation, NPAR, pp.113-120, 2004.

[Her98a] A. Hertzmann: Painterly Rendering with Curved Brush Strokes of Multiple Sizes. SIGGRAPH conference proceedings, pp.453-460, 1998.

[Kan07a]H. Kang, S. Lee, C. Chui. : Coherent Line Drawing, Proc. ACM Symposium on Non-photorealistic Animation and Rendering, pp.43-50, 2007.

[Kan08a] H. Kang, S. Lee : Shape-simplifying Image Abstraction, Computer Graphics Forum, Vol.27, No.7, pp.1773-1780, 2008.

[Kan09a] H. Kang, S. Lee, K.C. Chui :Flow Based Image Abstraction, IEEE transactions on visualization and computer graphics, Vol.15, No.1, pp.62-76, 2009.

[Kim96a] K.M Kim, C.S. Lee, E.J Lee, Y.H Ha : Color Image Quantization Using Weighted Distortion Measure Of HVS Color Activity. IEEE TRANSACTIONS ON IMAGE PROCESSING, Vol.3, pp.1035-1039, 1996.

[Kuw76a] M. Kuwahara, K. Hachimura, S. Eiho, M. Kinoshita: Processing of ri-angio cardio graphic images, Digital Processing of Biomedical images, pp.187–203, 1976.

[Kyp09a] J.E. Kyprianidis, H. Kang, J.Döllner : Image and Video Abstraction by Anisotropic Kuwahara Filtering, Computer Graphics Forum, Vol.28, No.7, 2009.

[Kyp11a] J.E. Kyprianidis, H. Kang : Image and Video Abstraction by Coherence-Enhancing Filtering, Computer Graphics Forum, Vol.32, No.8, pp.593-602, 2011.

[Lee10a] H. Lee, S.H. Seo, S.T. Ryoo, K.H. Yoon : Directional Texture Transfer. Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering (NPAR 2010), pp.43-48, 2010.

[Moj01a] A. Mojsilovic, E. Soljanin : Color Quantization and Processingby Fibonacci Lattices. IEEE transactions on image processing, Vol.10, No.11, pp.1712-1725, 2001.

[Ozd02a] D.Ozdemir, L.Akarun : A Fuzzy Algorithm for Color Quantization of Images, Pattern Recognition, Vol.35, No.8, pp.1785–1791, 2002.

[Pap07a] G. Papari, N. Petkov, P.Campisi: Artistic edge andcorner enhancing smoothing. IEEE Transactions on Image Processing,Vol.16, No.10, pp.2449-2462, 2007.

[Qia09a] W.H. Qian**,** D.Xu**,** G.Zheng: A NPR Technique of Abstraction Effects,2nd International Congress on Image and Signal Processing, Vol.1721, No.828, pp.1-5 2009.

[Tom98a] C. Tomasi, R. Manduchi: Bilateral filtering for gray and color images. In Proceedings international Conference onComputer Vision, pp. 839-846, 1998.

[Win06a] H. Winnemoller, C.S. Olsen, B.Gooch : Real-Time Video Abstraction. SIGGRAPH Conference Proceedings,pp.1221-1226, 2006.

[Zha95a] W. LI, Y.Q. ZHANG : Vector-Based Signal Processing and Quantization for Image and Video Compression. IEEE Processing, Vol.83, No.2, pp.317-335, 1995.

[Zha08a] H. Zhao, X. Jin, J. Shen, X. Mao, J. Feng : Real-Time Feature-aware Video Abstraction. The Visual Computer: International Journal of Computer Graphics, Vol24, No7-9, pp.727-747, 2008.

Contact Hardening Soft Shadows using Erosion

Andreas Klein
Munich University of Applied Sciences
Lothstrasse 64
80335 Munich, Germany
andreas.klein@hm.edu

Alfred Nischwitz
Munich University of Applied Sciences
Lothstrasse 64
80335 Munich, Germany
nischwitz@cs.hm.edu

Paul Obermeier
MBDA Deutschland GmbH
Hagenauer Forst 27
86529 Schrobenhausen, Germany
paul.obermeier@mbda-systems.de

ABSTRACT

In this paper, we present an image based method for computing contact hardening soft shadows by utilizing an erosion operator. Our method is based on shadow mapping and operates in screen space. By using object silhouettes in hard shadows, we estimate the penumbra size and scale an erosion operator to generate the penumbra areas. Furthermore, we present two solutions to generate the shadow factor for the penumbra areas. Our method works best for small penumbras and can be easily integrated into existing shadow mapping based applications.

Keywords

Shadow Mapping, Soft Shadows.

1 INTRODUCTION

Shadows are an important part for the human perception. They give clues about the spatial relationship and the form of objects. Real world shadows can be divided into an umbra and a penumbra. An umbra occurs when a light source is completely occluded and a penumbra when it is partially occluded.

In real-time rendering, a popular method to generate shadows is shadow mapping [Wil78a]. Shadow mapping assumes point light sources and thus, only hard shadows are produced. However, real world light sources are extended, and they generate penumbras, whose size often can be proportional to the light size and the receiver-blocker distance.

Current methods for generating contact hardening soft shadows are not suited for high shadow map resolutions [Lau07a], require a high amount of texture fetches [Fer05a] or the performance decreases with the number of shadow maps [Gum10a].

We present an algorithm to produce contact hardening soft shadows using an erosion operator. Our algorithm is an extension to shadow mapping and operates in screen space. Furthermore, it is suited for high shadow map resolutions as well as multiple shadow maps.

2 RELATED WORK

The rendering of soft shadows has been studied extensively over the last years. Therefore, we focus our review on publications closely related to our work. For an exhaustive survey on other methods see [Eis11a].

Percentage closer filtering (PCF) [Ree87a] is a popular method for generating soft shadows. The idea is to make multiple shadow comparisons within a user defined filter window. The shadowing factor is then built by averaging the result. To generate contact hardening soft shadows with PCF, Fernando [Fer05a] proposed percentage closer soft shadows (PCSS). He introduced a blocker search as a preprocessing step, where he sampled the shadow map to calculate an average blocker depth for each screen space pixel and approximated a penumbra width with a parallel planes approximation. Finally, he used the penumbra width to scale the PCF filter window.

Arvo et al. [Arv04a] estimated the penumbra regions by detecting the edges in hard shadows and propagating a visibility factor using a flood fill algorithm. Rong and Tan [Ron06a] accelerated this method using jump flood fill algorithms. Gumbau et al. [Gum10a] dilated a shadow map to replace the blocker search of PCSS. Furthermore, they replaced the PCF filtering with a separable Gaussian blur.

There are several approaches to calculate soft shadows in screen space. Robison and Shirley [Rob09a] used a screen space distance map to estimate a penumbra

width and blurred a hard shadow map with it. Hanjun and Huali [Han10a] developed an algorithm that propagates a shadow factor using erosion and dilation that is closely related to our work. However, we incorporate contact hardening soft shadows as well as penumbra anti-aliasing. Aguado and Montiel [Agu10a] presented an approach where a penumbra size is propagated using a mipmap flood fill and the penumbra is generated with a Gaussian filter in screen space. However, this approach produces light leaks which can be reduced by using multiple layers. MohammandBagher et al. [Moh10a] used a projected shadow map in screen space to estimate a penumbra size and to blur a hard shadow map

3 ALGORITHM OVERVIEW

Our algorithm computes the penumbra in screen space and is an extension to existing shadow mapping approaches. The algorithm proceeds as follows. First, we render hard shadows with a shadow mapping algorithm. Second, we detect edges in the hard shadows and store the blocker-receiver distance as well as the camera-receiver distance for the edge pixels. Now we can calculate the penumbra width since it is proportional to the blocker-receiver distance and indirect proportional to the camera-receiver distance. Next, we erode the edges with a filter kernel that is scaled according to the penumbra width and thus, estimating the inner and outer penumbra regions for contact hardening soft shadows. In order to generate the final penumbra, we propose two solutions. First, by filtering the shadow map in the penumbra regions with PCF and second, by directly calculating it during erosion.

Rendering Hard Shadows

The first step in the algorithm is to render hard shadows and auxiliary buffers. We assume that a shadow map has already been rendered. The scene is rendered from the observer and we perform a standard shadow comparison for each screen space pixel. We store a one in a screen space hard shadow buffer for each lit pixel and a zero for each shadowed pixel. Additionally, the depth difference between the blocker and receiver as well as the distance to the camera is stored. Furthermore, we store the diffuse color in a separate texture to calculate the final shading in the last pass.

Edge Detection

The second step is to estimate the penumbra regions by detecting the edges in the screen space hard shadow buffer. As the hard shadow buffer is a binary image, we can easily detect the edges with a 3 x 3 Laplacian filter, which needs five texture fetches:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

We calculate the penumbra width for each edge pixel by using the parallel planes approximation of Fernando [Fer05a]. Additionally, the kernel size is scaled based on the camera-receiver distance [Gum10a]:

$$\omega_{penumbra} = \frac{(d_{receiver} - d_{blocker})\omega_{light}}{d_{blocker} d_{observer}}$$

where ω_{light} is the light dimension. We store the penumbra width in the second texture channel.

Erosion

After the edges have been detected, they are eroded with a variable sized erosion filter[1]. We estimate the erosion by using a min-max mipmap [Isi06a, Dmi07a] (Figure 1). Recall that the edge texture stores a zero for each boundary pixel in the first channel and the penumbra size in the second channel. First we generate the min-max mipmap hierarchy for the edge texture by performing a min operation in the first channel and a max operation in the second texture channel. During erosion we calculate a maximum search radius for the given light dimension and the distance to the observer in order to find the closest edge, as the penumbra widths are only stored in the edge pixels. We choose a mipmap level based on the maximum search radius and query the min-max mipmap hierarchy. If the first channel contains no boundary pixel, we immediately terminate the erosion. Otherwise, we read the penumbra width from the second texture channel and choose again a mipmap level. Finally, we access the mipmap hierarchy in this mipmap level and test for boundary pixels. If a boundary pixel is found, we store the penumbra width in the result texture. Otherwise we discard the pixel.

Determine the Shadow Factor

In the final pass, we use PCF to determine the shadow factor for each pixel. We scale the PCF filter based on the penumbra width and combine the result with the hard shadow map rendered in the first pass.

4 ESTIMATE A SHADOW FACTOR WITH EROSION

A second solution is to estimate a shadow factor directly during erosion. In order to realize this idea, some changes in the algorithm are necessary.

As the screen space hard shadow buffer is rendered from the observer's viewpoint, objects may occlude shadowed areas and thus, the edge detector will produce edges which do not belong to penumbra regions. As Arvo et al. [Arv04a] pointed out, this issue can be solved by testing the shadow map for silhouettes on

[1] In terms of image processing this operation is an erosion, as the zeros in the edge texture are propagated.

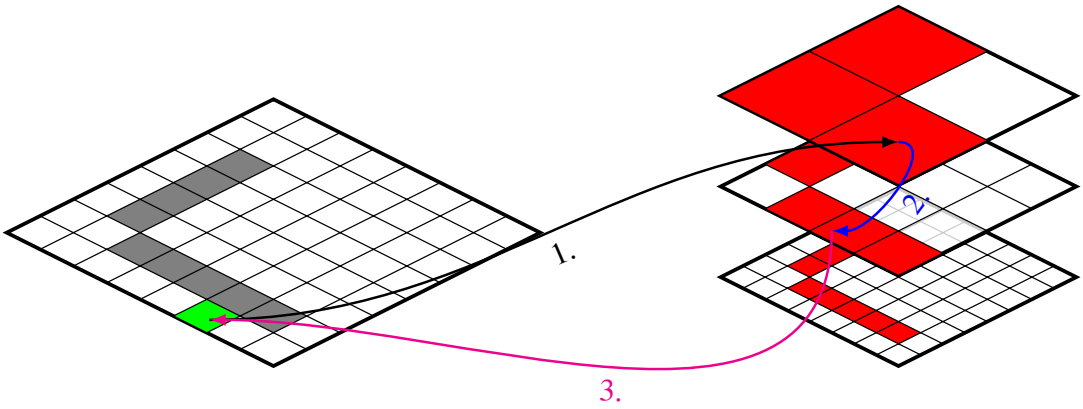


Figure 1: Erosion of the edges using a min-max mipmap. First, we choose mipmap level based on a maximum search radius and read the penumbra size from the max channel of the mipmap. Second, we calculate a mipmap level based on the penumbra size and read the edge value stored in min channel. Finally, we output the penumbra size if the edge value can be classified as a boundary pixel.

each detected edge pixel. We use the same 3 x 3 Laplacian filter and compare the result against a threshold. If the result is within the threshold, the edge pixel is valid and will be used in the next step. Otherwise, we discard it.

In order to compute the shadow factor, we implemented the variable sized erosion in a gathering approach. First, we determine a maximum kernel size and search for edge pixels within this area. If an edge pixel is found, we calculate its penumbra width and check, whether the current pixel is within its range. We continue until we found the edge pixel with the smallest distance to the current pixel. The shadow factor of the outer penumbra can then be directly calculated:

$$s_{outer} = \frac{\omega_{penumbra} - d_{min}}{2\omega_{penumbra}}$$

where d_{min} is the minimum distance to the edge and $\omega_{penumbra}$ the penumbra size. The inner penumbra is simply calculated with $s_{inner} = 1 - s_{outer}$.

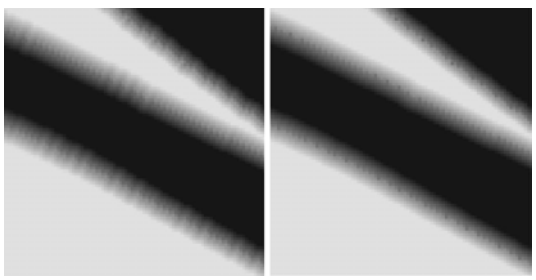


Figure 2: Left: Artifacts due to aliasing in hard shadows. Right: Result after the distance correction. Note that there are still some incorrect pixels at the transition from the outer to the inner penumbra.

Due to aliasing in the screen space hard shadow buffer, this method replicates the aliasing in the penumbra (Figure 2).

To increase the visual quality, we search for a best fit straight line by a least square method in a discrete environment around each edge pixel prior to the erosion and store the line parameters in an auxiliary texture. During erosion, we read the line parameters from the texture and calculate the vector v_{line} from the edge pixel's center to the line. Finally, we add v_{line} to the vector from the erosion point to the edge, calculate the distance and use it during erosion (Figure 3).

This compensates parts of the aliasing in the screen space hard shadow buffer and increases the visual quality (Figure 2). However, there are still some incorrect pixels at the transition from the outer to the inner penumbra. We will try to solve this issue in future work.

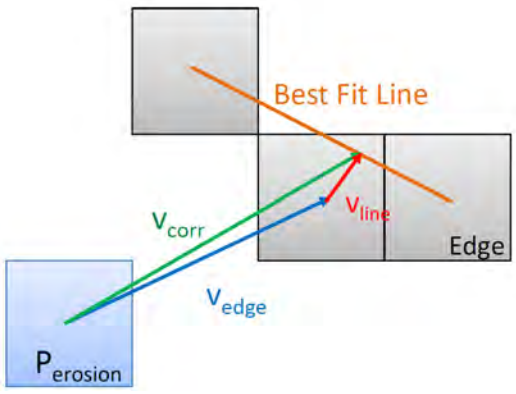


Figure 3: In order to improve the visual quality, we search for a best fit line and offset the vector to the edge with the vector from the edge to the line. Finally, we use the distance of the resulting vector during erosion.

5 RESULTS

We compared our algorithm against a PCSS implementation and a reference solution. This PCSS implementation uses a Poisson disk for the blocker search and PCF filtering. Both methods use 32 samples for the final PCF. The reference solution is realized by approximating the area light source with 512 point light sources. The screen resolution was 1920 x 1080 pixels and the shadow map size was 2048 x 2048. Figure 4 and 5 shows the resulting images and Table 1 the performance results. Table 2 shows the duration of the algorithm steps in the buddha dataset. The performance results were obtained on an Intel Xeon E5620 CPU with 2.4 GHz, 8 GB RAM and a NVIDIA GeForce GTX 680 graphics card with 2048 MB memory.

	Cactus (188K tris)	Hairball (2.88M tris)	Buddha (1.08M tris)
Erosion	1.62	12.56	5.77
PCSS 8	1.58	11.51	4.68
PCSS 32	1.60	11.89	5.11
PCSS 64	1.66	12.44	5.69

Table 1: Performance results in comparison with PCSS using 8, 32 and 64 blocker search samples. The screen resolution is 1080p.

Step	Time [ms]
Shadow Map	2.20
Hard Shadows	2.52
Edge Detection	0.31
Erosion	0.30
Shading	0.44

Table 2: Duration of the algorithm steps using the buddha dataset.

6 DISCUSSION

The bottleneck of PCSS [Fer05a] is the blocker search that is performed for each screen space pixel. Our motivation is to replace the blocker search per pixel by operating only on silhouettes of hard shadows. However, the rendering chain of our method is more complex. Thus, a speedup is only achieved when the blocker search is performed with 64 samples per pixel and the penumbra area is small. In contrast to PCSS, our method does not produce artifacts resulting from a small number of blocker search samples when rendering fine structured geometry, such as in Figure 4.

One possible issue in the method of [Gum10a] is that the algorithm operates on shadow maps. In contrast our algorithm operates on a screen space hard shadow buffer, which makes it attractive for applications with multiple shadow maps.

Compared to [Han10a] we incorporated variable sized penumbras and increased the visual quality by calculating the distance to a best fit straight line. Furthermore, we implemented a second solution for generating the shadow factor with a PCF filter, which results in a superior image quality.

Nevertheless, this technique has limitations. As our method is based on PCSS, it has the same limitations, such as overestimating the penumbra size. Furthermore, the erosion size is bounded and thus, we may miss relevant occluding information. This could result in visible artifacts. Due to mipmap erosion and the scaling of the penumbra width based on the distance to the camera, our method introduces aliasing when the camera is moved. Another limitation is that the visual quality is strongly dependent on the quality of the hard shadows. Consequently, aliasing reduction algorithms such as cascaded shadow mapping (CSM) [Eng06a] and light space perspective shadow maps (LiSPSM) [Wim04a] should be used.

7 CONCLUSIONS AND FUTURE WORK

We proposed a method for generating contact hardening soft shadows in screen space. As with all image based methods, this technique works best for small penumbras and can be used to extend shadow mapping based applications. Furthermore, we presented two solutions to generate a shadow factor for the penumbra. While the mipmap erosion is fast and produces results comparable to PCSS, the calculation of the shadow factor during erosion still produces some artifacts.

For future work, we wish to explore the possibility to replace the least square line fitting with a low pass filter and try to reduce the remaining artifacts.

8 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their useful comments. The work of A. Klein is funded by MBDA Deutschland GmbH.

9 REFERENCES

[Agu10a] Aguado, A. and Montiel, E. MipMapped Screen Space Soft Shadows. In GPU Pro 2. 2010

[Arv04a] Arvo J., Hirvikorpi M., Tyystjarvi J. Approximate Soft Shadows with an Image-Space Flood-Fill Algorithm. Computer Graphics Forum 23, 271-279, 2004.

[Dmi07a] Dmitriev K., Uralsky Y. Soft shadows using hierarchical min-max shadowmap. GDC 2007, 2007.

[Eis11a] Eisemann E., Schwarz M., Assarsson U., Wimmer M. Real-Time Shadows, Taylor & Francis, 2011.

[Eng06a] Engel W. Cascaded shadow maps. In Shader X5, Engel W., (Ed.). Chares River Media, 197-206, 2006.

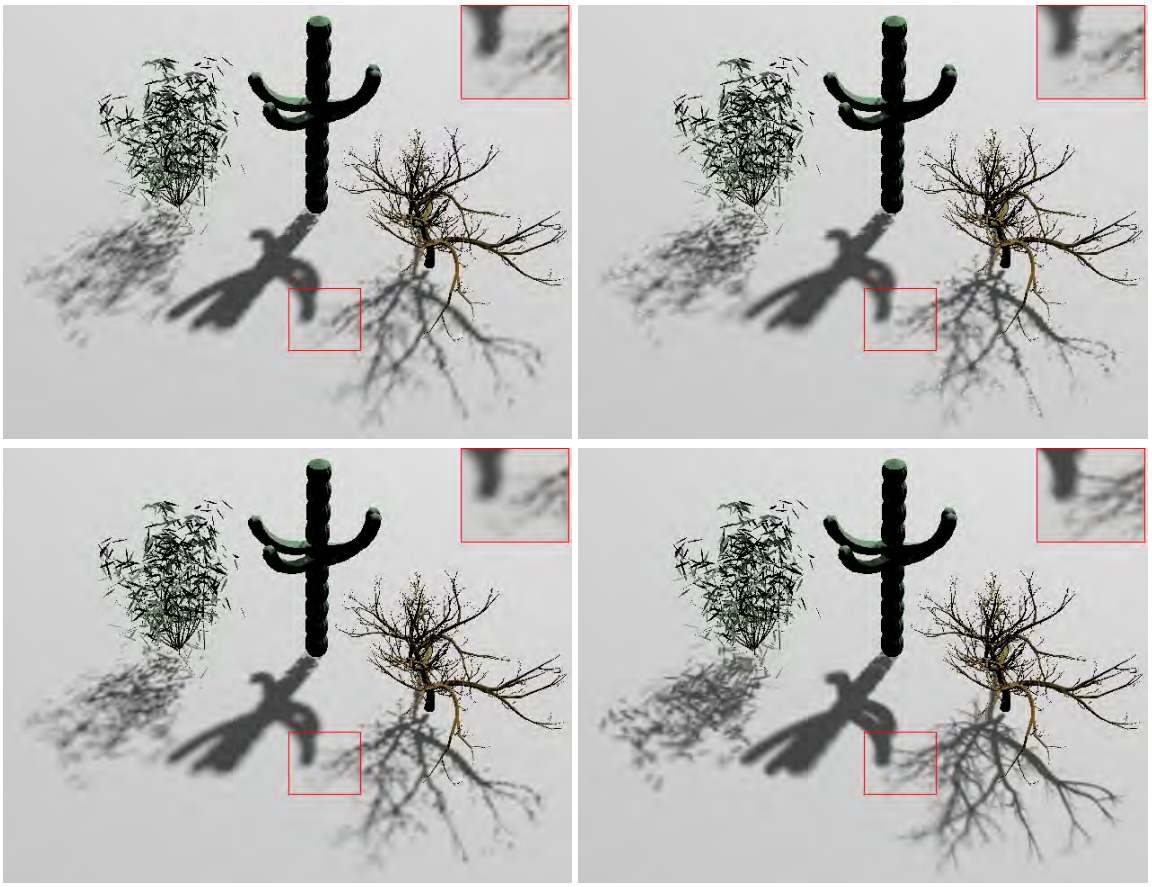


Figure 4: Visual results in the cactus dataset. Top Left: Our algorithm. Top Right: PCSS with 8 blocker search samples. Notice the artifacts resulting from missed blockers due to the small number of blocker samples. Bottom Left: PCSS with 64 blocker search samples. Bottom Right: reference solution.

[Fer05a] Fernando R. Percentage-Closer Soft Shadows. ACM SIGGRAPH 2005 Sketches, 2005.

[Gum10a] Gumbau J., Chover M., Sbert M. Screen space soft shadows. In GPU Pro - Advanced Rendering Techniques, Engel W., (Ed.). A.K. Peters, 2010.

[Han10a] Hanjun J., Huali S. Rendering fake soft shadows based on the erosion and dilation. 2nd International Conference on Computer Engineering and Technology, 234-236, 2010.

[Isi06a] Isidoro J. R. Shadow Mapping GPU-based Tips and Techniques. GDC 2006, 2006.

[Lau07a] Lauritzen A. Summed-area variance shadow maps. In GPU Gems 3, Nguyen H., (Ed.). Addison-Wesley, 157-182, 2007

[Moh10a] MohammadBagher M., Kautz J., Holzschuch N. and Soler, C. Screen-space percentage-closer soft shadows. ACM SIGGRAPH 2010 Posters, 2010.

[Ree87a] Reeves W. T., Salesin D. H., Cook R. L. Rendering antialiased shadows with depth maps. In Conf.proc SIGGRAPH '87, ACM, 283-291, 1987.

[Rob09a] Robison A. and Shirley P. Image space gathering. In Conf.proc. High Performance Graphics 2009, 91-98, 2009.

[Ron06a] Rong G., Tan T.-S. Utilizing jump flooding in image-based soft shadows. In Conf.proc. VRST '06, ACM, 173-180, 2006.

[Wil78a] Williams L. Casting curved shadows on curved surfaces. In Conf.proc. SIGGRAPH '78, ACM, 270-274, 1978.

[Wim04a] Wimmer M., Scherzer D., Purgathofer W. Light space perspective shadow maps. In Conf.Proc. Eurographics Symposium on Rendering, 2004.

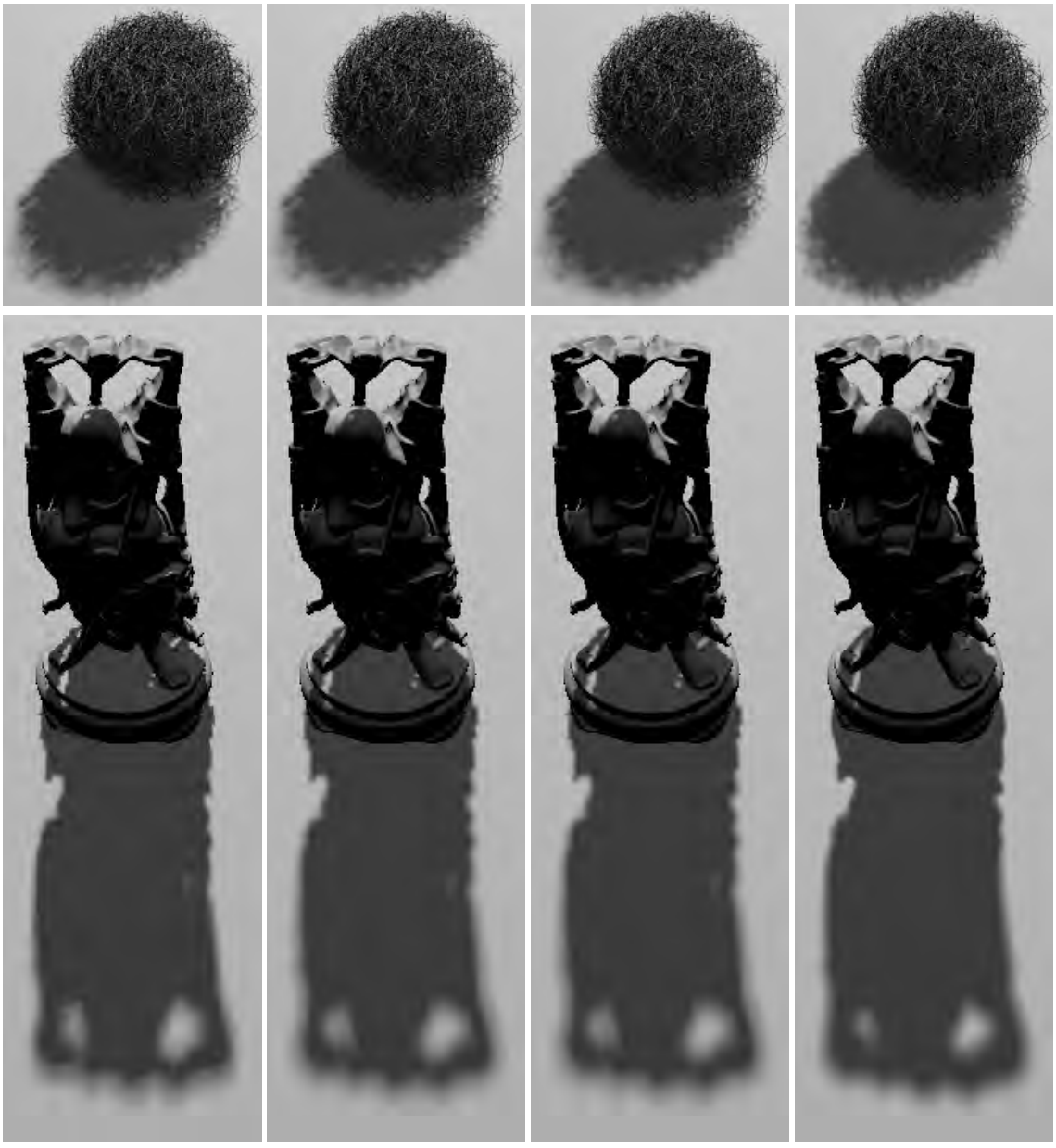


Figure 5: Resulting shadows from the hairball and buddha datasets. Note that the shadow softness increases with the blocker-receiver distance. From left to right: Our method, PCSS with 32 blocker search samples, PCSS with 64 blocker search samples and reference solution.

Diffusion-based parametrization of surfaces on 3D-meshes

Martin Schmidt
Philipps-University of Marburg
Hans-Meerwein-Str. 6
Germany, 35032 Marburg
schmidtma@informatik.uni-marburg.de

Michael Guthe
Philipps-University of Marburg
Hans-Meerwein-Str. 6
Germany, 35032 Marburg
guthe@informatik.uni-marburg.de

Volker Blanz
University of Siegen
Hölderlinstr. 3
57076 Siegen
blanz@informatik.uni-siegen.de

ABSTRACT

This work presents a new approach towards parametrization of three-dimensional wireframe models. The method is derived from a specific phenomenon of cellular development in nature. It recreates the effect of diffusion of messengers through tissue, which leads to the formation of extremities and other anatomical structures depending on the position on the tissue surface. This process of diffusion on the surface is analyzed and simplified for usage as a parametrization of mesh surfaces. The presented approach uses the similarity of wireframe meshes and graphs in order to carry out the mechanism of diffusion. For this it implements a specialized algorithm based on Dijkstra's algorithm for finding the shortest paths.
The results of this mechanism are saved and organized in a binary tree structure, which allows for simple and efficient construction of correspondence between two distinct meshes. The paper concludes with an outlook on possibilities of further development and enhancements of the approach.

Keywords

mesh, parametrization, diffusion, geometric analysis

1 INTRODUCTION

Whenever computers analyze or display visual objects, they need to represent these objects in a suitable mathematical form that is appropriate to the processing tasks. The typical representation of an object is a polygon mesh, but for many purposes such as texturing, object retrieval, shape comparisons, differential geometry or for computing point-to-point correspondence between pairs of objects, it is important to describe shapes as parameterized surfaces. Hence, the purpose of this paper is to introduce a new approach in parametrization and its evaluation.

1.1 Parametrization

In the following, we give a short introduction on the topic of parametrization, its applications and related work. This is concluded by the analysis of the distinction of our approach in contrast to previous work.

Basics

An important aspect of 3D shape processing and analysis is the connection of objects with each other. By a correct link between two objects, algorithms can transfer knowledge and properties from one object to the other. It is also possible to describe the similarity between these objects. This link is generated by a surface parametrization that is consistent across different objects.

A parametrization of surfaces is defined as a mapping from a parameter domain onto the surface. This is also called *Surface-to-Surface-Mapping*, if the parameter domain is a surface itself. The parametrization of a parameter domain maps each point on the domain onto a certain point of the surface.

Bijectivity is an important property of parametrizations, as many applications rely on a complete cover of the originating surface without introducing ambiguity. That means each point on the surface maps to exactly one point on the parameter domain and vice versa.

Possible applications in computer graphics

There are several possible applications for parametrizations in computer graphics. The three most important applications are briefly described in the following.

1.1.1 Transfer of detail:

One of the first times a parametrization of objects became necessary was the application of *texture mapping* in rendering (see Figure 1. As a part of the rendering of a scene, texture mapping increases the detail of the surfaces of objects by drawing a pregenerated image on the surface.

Figure 1: Texture mapping

Further improvements of this approach include *Normal Mapping*, which transfers the shading of a high quality mesh consisting of thousands of polygons to a mesh with reduced polygon count to increase the detail on the latter mesh without decreasing render speed. More approaches like this are *Bump Mapping* and *Displacement Mapping*, which – similar to *Normal Mapping* – also apply more detail onto a mesh without significant impact on the render time.

1.1.2 Remeshing:

Polygonal meshes are created by several methods like scanning with lasers and modeling by hand in a special software. This leads to meshes of different resolution and different surface generation techniques (e.g. triangles, quads, mixes of both). Sometimes an application only allows for a certain kind of triangulation and resolution.

This is where *Remeshing* becomes important. Remeshing parametrizes a mesh and then maps a regular and desired triangulation on the parameter domain to retriangulate the original mesh [13]. The application of subdivision on the parameter domain can also lead to good results in regard to desired mesh quality [17], [14].

1.1.3 Correspondence:

If two meshes should be analyzed, it is sometimes desired to link those meshes to each other by means of correspondence. The correspondence of two meshes means that the relationship between a region A on the first mesh and a region B on the second mesh is known. This can be used to transfer details from one mesh to the other.

To create this link, both meshes need to be mapped to the same representation. By using a bijective parametrization, an algorithm can map each point on the first mesh to the corresponding point on the second mesh simply by choosing the same parameter values.

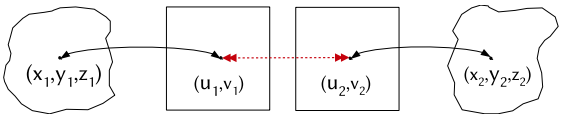


Figure 2: Correspondence between two surfaces

1.2 Related Work

The increasing requirements for parametrizations on surfaces have led to the development of different approaches with different pros and cons. The following section summarizes these attributes.

Criteria for the evaluation of parametrizations

The attributes, by which parametrizations can be measured regarding their potential application, are:

- the degree of distortion
- the aspect of bijectivity
- the limitation on certain mesh types

Distortions in the parameter domain are a direct result of parametrization and can be of different type. It has been proven by differential geometry that there is no distance-preserving parametrization for generic surfaces [8]. The distortion can be minimized, but not fully prevented.

Possible types of distortion are the distortion of distance – an irregular distribution of parameter values in one dimension – and angular distortion. Type and degree of distortion are important criteria in judging methods of parametrization. It is often dependent on the specific application which methods are more suitable than others. This leads to compromises almost every time since no parametrization is completely free of distortion.

The methods of parametrization also show different bijective behavior. This can be separated into global and local bijectivity. Global bijectivity is maintained over the whole mesh, while local bijectivity is given only for local regions on the surface. Not every parametrization leads to bijectivity of one of these kinds.

The third criterion is the limitation on certain mesh types. Some parametrizations need convex meshes, while other approaches can also use more complex meshes.

Approaches

Perhaps the oldest method of parametrization was developed by William Tutte in 1963 [27]. Tutte used *graph embedding* as the basis of the approach, which led to a bijective parametrization with distortion in distance and angular aspects. Using the same approach,

two algorithms developed by Floater show similar behavior, but at the same time reduce angular distortion [10], [11].

Eck et al. use a parametrization for remeshing at different resolutions [9]. This method is based on harmonic maps and therefore preserves angular dimensions.

DCP is a different parametrization developed by Desbrun et al. [6] and combines the already known Dirichlet-energy [20] with a new quadratic Chi-energy E_χ, which describes the inner angles of triangles. This method is not bijective in every situation, but can be used without limitations (e.g. only convex meshes).

Implementing the *Least-Squares*-method, the parametrization called LSCM preserves the orientation of each triangle and angle. It is independent of resolution but cannot guarantee bijectivity for every mesh [18].

Linear parametrizations like those mentioned above tend to create an increased distance-based distortion on meshes with sharp slopes. Using non-linear parametrizations helps to reduce these distortions. An example is MIPS [15], which divides the mesh into several linear maps. A special functional reduces the distortion map by map and creates a parametrization which is bijective and can be used without limitation.

The parametrization ABF differs from the mentioned methods, as it does no work on the vertices, but on the angles of the triangles [22]. It reduces angular distortion and shows local bijectivity. A variation of the algorithm called ABF++ increases calculation speed and stability on large meshes.

Kharevych et al. adopt a similar approach by defining circumcircles on every vertex [16]. Cutting circumcircles define the angles between vertices, which are then minimized. This approach works best on Delaunay-triangulations [5].

The introduced parametrizations are primarily based on mathematical ideas and concepts from computer science. These are the topics of differential geometry, topology and graph theory, which are combined to represent a mesh in parameter domain.

In the past, several approaches to carry concepts over from natural processes to computer science were successful and lead to groundbreaking and novel methods, for example genetic algorithms, routing algorithms and graph algorithms [19, 7, 1]. Other approaches that use similar diffusion-based processes called *reaction-diffusion* create textures automatically [26, 28].

In the topic of parametrizations, this transfer is yet to be made.

1.3 Motivation

This work applies insights from biology and chemistry to the problem of consistent surface parametrization. Diffusion helps cellular development and differentiation by giving hints on the position in the organism to individual cells. The aim of this paper is to show that the natural processes of diffusion can help in the development of algorithms for parametrizations.

The proposed method guarantees local bijectivity on convex mesh parts, which are constructed from the original mesh. The parametrization retains the proportions of distance of the mesh parts. Because the mesh is viewed as a graph, already available and highly optimized algorithms from graph theory can be used to achieve an efficient and stable parametrization.

In the following, the course of the paper is to present an introduction to physical and biological diffusion, followed by evaluation of algorithms which can be appropriate for serving as a basis for further development. Later on, the modifications to the chosen algorithm are explained. The paper concludes with a discussion and and an outlook on possibilities for further research.

2 DIFFUSION

The basis of our approach originates from morphogenesis. Morphogenesis controls the differentiation and development of cells in multicellular organisms to organs and extremities, and it produces patterns on skin, fur or shells of animals. Well-directed flow and diffusion of activators through the tissue leads to specific development of the stem cells depending on the structure they are going to form.

The gradients of concentrations of specific substances form a metric in the tissue and on the surface. On different shapes of the same type, these gradients and thus the induced metric are similar, so they describe objects regardless of position in space, scale, orientation and resolution of the mesh.

2.1 Physical diffusion

Diffusion occures whenever particles – for example atoms, molecules or charge carriers – are aggregated in a carrier medium and there exists a concentration gradient between these particles. The reason for this movement is called pedesis or Brownian motion [3]. The atoms and molecules are in an undirected motion, depending on temperature: Due to collisions, their direction changes randomly over time. If there is a concentration gradient, there will be an overall net motion along this direction, which forms the diffusion process described by the following equation:

$$\frac{\delta u}{\delta t} = D \cdot \Delta u = D \cdot \operatorname{div}(\operatorname{grad} u) \qquad (1)$$

2.2 Diffusion in developmental biology

Diffusion plays an important role in biological processes. By diffusion through membranes cells are supplied with nutrients and metabolic waste products are removed. Patterns on the skin or fur are also controlled by diffusion of messengers.

This is described in detail by Gierer and Meinhardt, who present a model which depicts the formation of structures in biological cell structures [12]. The interaction of two messengers – the so called activator/inhibitor pair – play a major role in position dependent pattern formation. Both agents diffuse through the tissue and lead to different concentrations depending on position. This process – called morphogenesis – has been described by Turing [25] and was specified further by Gierer and Meinhardt. A gradient of concentration values runs between the cells producing the activator and between the cells producing the inhibitor. The resulting patterns of cell differentiation are aligned along this axis.

One special form of this mechanism is observable in the polyps of the genus *Hydra*. Their heads and feet are shaped depending on the concentration of the activator [12]. Gierer and Meinhardt showed that the diffusion of the morphogenes in Hydra takes a gradual course. This gradient of source density gives orientation in the tissue over the longitudinal axis of the animal's body. The gradient serves as an indicator of relative position within the animal.

2.3 Formulation of the idea

Several criteria influence the parametrization and construction of correspondence, depending on the geometry and quality of the mesh. These are:

- Scale of the mesh
- Position of the mesh relative to the coordinate system
- Rotation of the mesh
- Internal geometry of the mesh (for correspondence a rough similarity is sufficient)

There may be considerable variation in these parameters and properties, which makes many surface analysis problems challenging. In many cases, surface parametrizations help to find more simple and efficient solutions.

Our diffusion-based parametrization is inspired by the gradient of source density, which defines a polar orientation. We will ignore most of the details of the biological mechanism and develop the idea towards parametrization of surface patches.

2.4 Diffusion as a description of surfaces

To successfully compute the diffusion on the surface, the physical and biological model needs to be simplified. The first simplification is to consider the surface as a two-dimensional carrier medium and discarding any effects of diffusion into depth. Both complexity and computational costs are decreased due to the fact that only two dimensions are considered.

We assume a dynamic equilibrium where a substance is produced in a source point or a line-shaped set of sources, travels along the surface due to diffusion and is diluted, washed away, absorbed or deactivated chemically over time. Therefore, in our simplified model, the concentration of the substance is proportional to the distance from the source.

To obtain relative distances on the surface, independent of the scale of the object, we introduce pairs of antagonist substances that have concentrations d_1 and d_2 and are defined on each vertex of the mesh. To calculate the relative distance between each of the reference points, the following equation is used:

$$\hat{d} = \frac{d_1}{d_1 + d_2} \tag{2}$$

This formula makes sure that $\hat{d}$ takes the value 0 on the source points of the first substance and value 1 on the source points of the second substance. It then increases smoothly in between. The possible values of distance $\hat{d}$ are clamped to the interval $[0,1]$.

This operation is the necessary first step in parametrizing the whole mesh. To get an even distribution of the parameters, sets of points form the source points. This leads to a smooth and near parallel distribution of the isolines on the surface (see Figure 3).

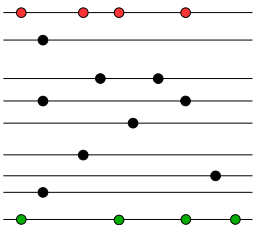


Figure 3: An ideal distribution of isolines of $\hat{d}$ is obtained if sources for d_1 (*red*) and d_2 (*green*) are not only single points, but sets of points along the edges

For a unique identification of each point a second axis in parameter distribution is needed (see Figure 4). If this second axis is orthogonal to the first direction of diffusion, two linearly independent parameters result and are able to describe every point on the surface by a pair of the interval $[0,1]^2$. We call these two axes the gradients of diffusion ∇u and ∇v. The parameters are called u and v and form pairs (u_i, v_i) for each point P_i on the surface.

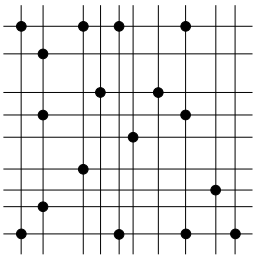


Figure 4: A 2D parametrization (u,v) is obtained by two separate diffusion processes, one along the horizontal and one along the vertical direction, with two variables $\hat{d}_u$, $\hat{d}_v$

2.5 Organization of the parameter values in a kd-tree

By simply saving the parameter values to each point, one can easily retrieve the tuple (u,v) for every given point. This is done in a linear array, which can be accessed in $\mathcal{O}(1)$. For an adequate use the other way around is also important, therefore it is necessary to have an efficient data structure for finding a point corresponding to a given value. Different data structures exist, which help to find a given value in a set of points quickly without comparing each point to the search term. In our work we chose the kd-tree [2], which scales well with a certain number of points. The complexity of searching for a value lies in $\mathcal{O}(\log n)$, where n is the number of vertices of the mesh.

2.6 Viewing surfaces as a discrete point set

Polygonal surfaces are usually intended to be approximate models of smooth surfaces such as geometrical primitives (spheres), vehicles, human faces or characters. The higher the number of polygons is, the better the approximation of the surface can be. Because of this approximation it is difficult to model the process of diffusion, as it runs through a continuous substance. The triangulated surface has no continuous nature, but instead has gaps between discrete points.

The algorithm to solve this problem needs to simulate the continuous progression of the diffusion along the surface with special regard to the distance between the points. One algorithm with these characteristic is Dijkstra's algorithm for shortest paths. With a given starting point, Dijkstra's algorithm assigns each following point the shortest distance to the starting point. It propagates the distance values iteratively in a process that is similar to diffusion.

We modified the algorithm in such a way that it takes a complete set of points to start instead of a single point. After a successful pass, each distance d_1 between point and start is measured. A second pass in the reverse direction leads to the second value d_2 of diffusion, which is used in order to determine the value of relative distance $\hat{d}$ in the calculation.

We deliberately decided to propagate the values along edges. Alternatively one could choose the geodesics, i.e. direct paths along the surface which are not constrained to points in the set. Geodesics, however, do not respect the structure of the mesh. Interpolation is required to calculate the geodesic path through the polygons. Because of this computational increase the idea of geodesics had been discarded.

2.7 Segmentation

Whether global bijectivity of the parametrization can be achieved at all depends on the topology of a mesh. A mesh which is topologically equivalent to a disc can be mapped to a parameter domain in the plane. These simple meshes do not need to be partitioned and easily achieve global bijectivity. Other meshes with more complex topology cannot be mapped to a plane and therefore violate global bijectivity.

Consider the simple example of a sphere. From topology, we know that we cannot find a homeomorphism from parameter space $[0,1]^2$ to the sphere, so we expect singularities and ambiguities if we apply our algorithm to the entire sphere. For the first parameter u and its relative distance function $\hat{d}$, consider a line from pole to pole (half of a great circle) as a source for d_1, and the other half of the same great circle as source for d_2. This defines a diffusion which spreads over both hemispheres. However, both poles will be singularities because they are sources of both d_1 and d_2. On the other hand, if we use two opposite points as sources for d_1 and d_2, respectively, we violate the uniqueness criterion as soon as both parameters u and v are computed: As shown in Figure 5, two arbitrary isolines of each diffusion cintersect twice, so these intersection points obtain the same parameter tuple (u,v).

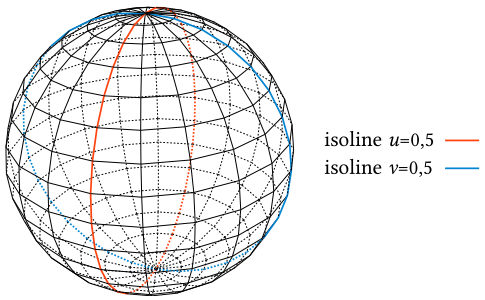


Figure 5: Ambiguity on certain points on meshes that are not homeomorphic to a disc.

The solution to this undesired effect is the segmentation of the mesh into hemispheres. Two passes parametrize each hemisphere separately. This guarantees local bijecitivity on each partial mesh. If the segmentation of the mesh is adequate, the whole mesh can be partitioned in topological discs with guaranteed uniqueness.

Global bijectivity is not given and therefore the direct mapping of (u,v)-tuples from two meshes is not possible. A mechanism is required to assign regions of different meshes to the same or similar areas, which introduces a new level of hierarchy.

The segmentation must lead to reproducible regions. Only then is the parametrization useful, particularly in regard to the correspondence between meshes. Several criteria define the usefulness of the segmentation. Besides the reproducibility these are the number of segmented regions, similar segmentation on similar meshes and interaction of the user to mark logical areas.

To achieve a 1 : 1 connectivity between meshes, the segmentation must separate both mesh A and mesh B to each set of regions $r_{A_0}, r_{A_1}, .., r_{A_n}$ and $r_{B_0}, r_{B_1}, .., r_{B_n}$. Each vertex must be assigned to a region. Both sets of regions can be related to each other if both meshes have the same number of regions. To create a meaningful correspondence, the regions must map to similar parts on the meshes in the first place. This means that the segmentation must follow a fixed orientation over the mesh, which also adds to the reproducibility.

Meshes from different classes of objects should segment to semantic similarities of the same type. The algorithm should react on the user's input, which denominates the points of interest regarding the logical areas. This simplifies the finding of correspondence between meshes and enhances quality.

Existing approaches for reproducible segmentations lack the possibilities to work on a predefined number of regions and only respect the user's input marginally. Therefore, we chose to implement a simple, but efficient approach that gives the user the freedom of choice in segmenting the mesh (see Figure 6).

The idea is that the user paints the regions on the surface manually with an interactive tool. This defines the segments of the patchwise parametrization. Moreover, and this is the core idea of the user interface, the boundaries of the segments are a natural choice of source-sets for the diffusion algorithm. Depending on the label of the adjacent region, a boundary vertex will be a source point of d_1 or d_2 for parameter u or v, respectively. The goal of the segmentation is to produce patches with closed and connected boundaries that can be divided into 4 parts, similar to a rectangle.

Those parts which form the ends of extremities (e.g. hands and feet in Figure 6) of the depicted mesh need special treatment. They could be seen as single regions, because they fulfill the topological criterion. The problem of this naive approach is the nature of their edges. These edges overly stretch during the mapping to the plane (as seen in 7). This leads to increased angular distortion, with higher extremes in parts with longer stretched edges.

Figure 6: Segmentation example

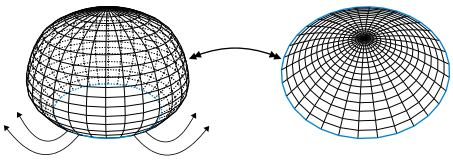


Figure 7: Edge stretching and angular distortion

Separating such meshes into two halfs helps preventing these unfavorable results. By splitting the meshes in question before mapping them to the plane a lower distortion is achieved, as the change of length of the edges is reduced.

After choosing the regions carefully to avoid bad segmentation, the next step is the selection of starting and ending edges. As mentioned before, the process of diffusion needs the four edges of the surface to be parametrized. The edges resemble the sets of starting and ending points for the modified Dijkstra. Because the diffusion is not limited to four-sided surfaces, it must be possible to choose four logical edges on arbitrary convex surfaces, leading to edges $E_{u_1}, E_{u_2}, E_{v_1}, E_{v_2}$. The finding of suitable edges can be based on the segmentation into regions, as the borders between two regions already form suitable edges.

In the case that there is only one continuous border between two regions, the edge is simply the overlapping part of both regions. In order to not include a set of points in two edges at once, the method can choose to include or exclude these vertices. This discrimination is necessary if a point takes part in both regions as seen in Figure 8.

A user who is familiar with the process of segmentation can handle it very quickly. The segmentation in Figure 6 did not take longer than 10 minutes with the interactive, mouse-based tool.

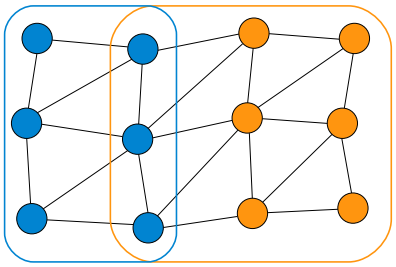


Figure 8: Overlapping of vertices in two regions

2.8 Process of diffusion

The diffusion processes each region on the mesh separately. This is done by separation and transformation of the whole mesh into partial graphs. As all regions are combined to the original mesh, the partial graphs of each parametrized region form the whole graph that represents the mesh.

The construction of partial graphs shifts the process of diffusion away from the original mesh. Each partial graph leads to an own uv-map, which in turn is independent to the uv-maps of other partial graphs, thus solving the global bijectivity problem. But as shown later, the local bijectivity on each partial graph is given.

The modified Dijkstra algorithm is initialized just like the original algorithm. Each vertex that does not belong to the starting points is assigned $d_i = \infty$. Unlike the original Dijkstra's algorithm, which starts from a single vertex, it is here the initial starting (source) set that is initialized as $d_{i_S} = 0$. After inserting each point of the whole partial graph into a *min-heap*, the algorithm can start.

Taking the first element out of the min-heap gives the point with the minimal value of diffusion $d_{i_{min}}$. In the first iterations this will be the whole set of starting points, but later on this will govern all points in the whole partial graph. Each extracted point is treated in exactly the same way. Every edge of this vertex will be relaxed, using the following equation 3.

$$d_B = \begin{cases} d_A + w(A,B), & \text{if } d_A + w(A,B) < d_B \\ d_B, & \text{otherwise} \end{cases} \tag{3}$$

The function $w(A,B)$ is the weighting function which returns the length of the edge between vertices A and B. Vertex A is the vertex that is currently being processed and vertex B is the target of the edge that the algorithm actually relaxing. Therefore, A stays the same for the vertex that is processed and B changes to each of the adjacent vertices during relaxation of the adjacent edges.

This relaxation leads to updated values of diffusion in each adjacent vertex. By updating the value, the process of diffusion iteratively spreads over the whole graph, until reaching its end (see Figure 9 for an illustration). Since the relaxation affects the value of diffusion, which in turn is the key to the min-heap, the residing vertices are sorted after each update to reflect the correct sequence of increasing key values.

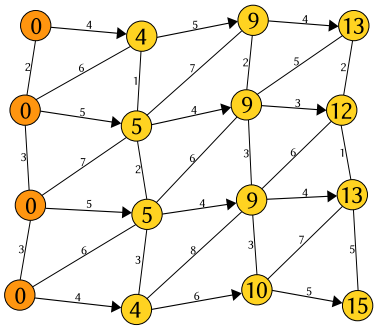


Figure 9: The process of diffusion illustrated with a simplified mesh.

As soon as the min-heap is empty, the process of diffusion finishes. At this point, each vertex holds its distance to the set of starting points. This distance is the path to the closest vertex in the set of starting points. This algorithm is run twice, once for d_1 and once for d_2. From these, $\hat{d}$ is computed, which is equivalent to the parameter u. With another pair of source sets, the parameter v is computed in the same way.

After the completion of the second two-pass, each vertex holds the values d_{u_1}, d_{u_2}, d_{v_1} and d_{v_2}. These values are the direct values of distance to their corresponding starting edge. By inserting these into the two equations 4, we normalize the values into the interval $[0,1]$:

$$\hat{d}_u = \frac{d_{u_1}}{d_{u_1} + d_{u_2}} \qquad \hat{d}_v = \frac{d_{v_1}}{d_{v_1} + d_{v_2}} \tag{4}$$

These values of diffusion $\hat{d}_u$ and $\hat{d}_v$ are then saved in the kd-tree. By combining the kd-tree with a simple linear array, we get a uv-map that supports fast retrieval of (u,v)-tuples for a given vertex and also grants an efficient search for a vertex, which lies as close as possible to a given (u,v).

3 RESULTS

We implemented the modified Dijkstra's algorithm. We parametrized a scan from a human and a cow and manually developed a segmentation of the meshes. The human mesh (see Figure 10) containes about 11k vertices with 3 to 5 edges adjacent to each vertex. The cow (see Figure 11) is made up of ca. 3.1k vertices, also connected to neighbors with 3 to 5 edges. The parametrizations were done in 68 ms for the cow and 224 ms for the human mesh on a Pentium i7-2600 3.40 GHz.

The parametrization of the whole mesh took place as a sequence of parametrizations on the list of partial

meshes after the segmentation. Like the original algorithm by Dijkstra our modified approach has a complexity lying in $\mathcal{O}(|E|log|V|)$, where $|E|$ is the number of edges in the mesh and $|V|$ is the number of vertices in the mesh. We then applied a grid on the mesh, using the parametrized values as texture coordinates (see Figure 10).

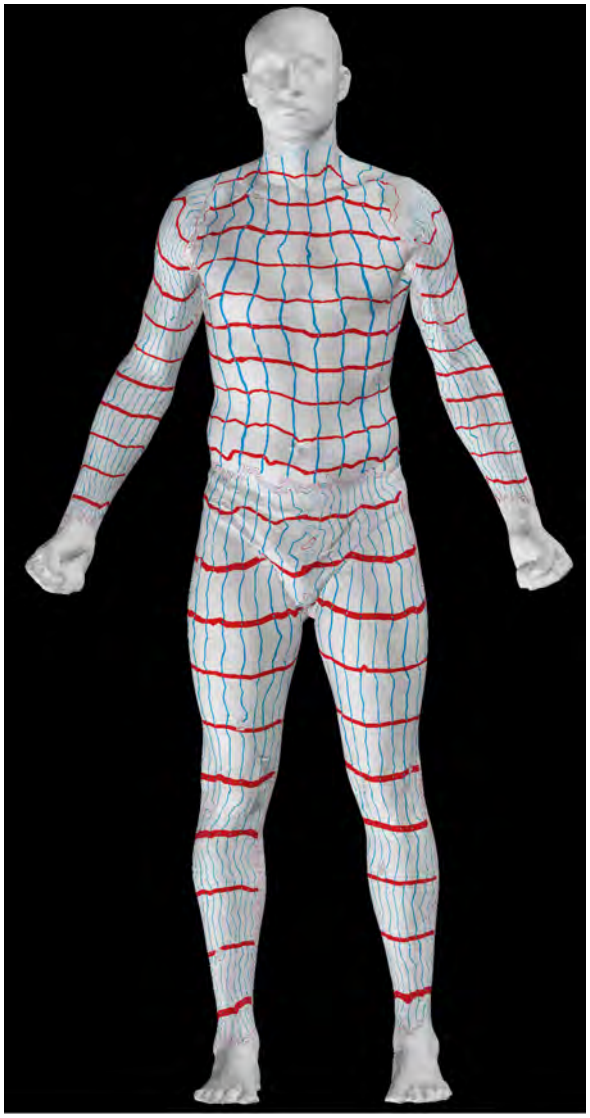


Figure 10: The parametrization visualized by a grid

From these pictures we can judge the quality of the parametrization. The distribution depends on the mesh structure and it is clearly visible that this leads to an increased jitter of parameter values.

On the large areas like chest and back of the human mesh (Figure 10) there is less distortion than on structurally smaller areas. These areas (e.g. the arms and legs) show a ratio between horizontal and vertical distribution which is not aspect-preserving.

The different length of the edges in a mesh is directly visible in the grid. For example, the edges on the side of the cow are longer than the edges on other parts of the mesh. The apparent gap between the isolines shows that the edges on this side are almost double in length. Right behind the right shoulder of the cow there is a loop that continues on the bottom side of the mesh. This loop is part of the isoline that moved downwards due to the larger distance between the vertices on the side.

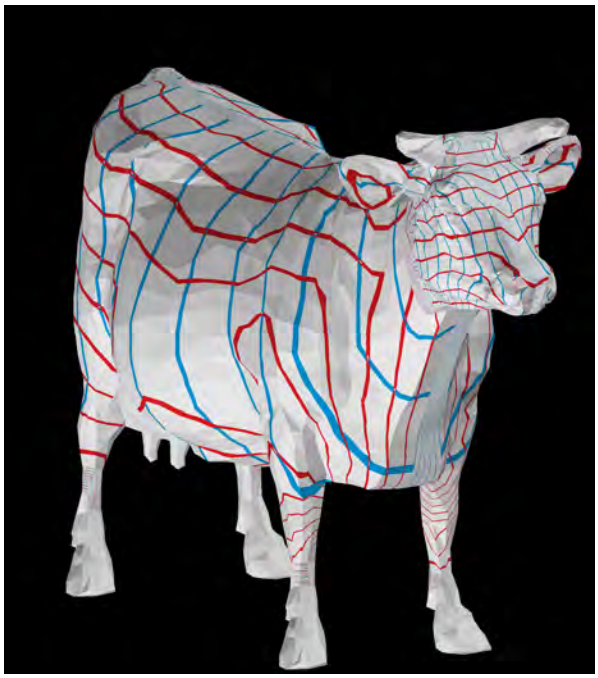


Figure 11: Parametrization of another object

An important aspect is the examination of the boundaries between the regions. They show irregularities which probably stem from the different parameter values that overlap on these vertices. Regardless of which set of (u,v)-values is chosen, there will always be a residual distortion, even though the algorithm contains mechanisms that guarantee a continuous parametrization across boundaries of segments: Consider a segment pattern that looks like a distorted rectangular grid on the 3D surface, and a boundary segment S between two patches A and B. Let the parameter u be 1.0 along the boundary in A, and 0.0 in B, while v varies from 0.0 to 1.0 continuously. Then v is determined by the boundaries (source sets) adjacent to S in A and in B. If these adjacent boundaries in A and B fit together as one continuous curve, like in a rectangular grid, the parameter v on S will be the same in A and B, so the patch-wise global parametrization is continuous across patch boundaries.

3.1 Comparison

Based on Alla Sheffer's work [24], we compared our parametrization with other approaches. Table 1 shows various parametrizations [24]. We added our parametrization in the last line.

The images show clearly that the algorithm does hardly prevent distortion. Especially in parts with complex curvature (e.g. the head and breast of the cow) angular distortion increases. Deficiencies on the mesh (e.g. different edge lengths, jumps, doubled vertices) further increase the distortion. Adding the evaluation of an error metric before the parametrization can value the quality that can be expected.

Especially the uniform, harmonic and mean-value weighted parametrizations [27, 9, 11] show similarities between different patches on the surface, e.g. the breast region of the cow. Algorithms like DCP and LSCM have similar problems preserving areas and distances, but are better in preserving angles than our algorithm. This is because they sacrifice the distance preservation in favor of minimized angular distortion. Our algorithm has problems with angular distortion in specific regions.

Singularities like the ears of the cow show cycles that are direct result of the diffusion process. Other parametrizations handle such singularities more graceful. Circle patterns [16] and stretch minimizing [21] approaches excel at these parts.

4 CONCLUSION AND FUTURE WORK

In this paper we showed the development of a new approach in parametrization, inspired from nature. Our approach can lead to a patchwise bijective parametrization, which concentrates on local bijectivity. User interaction makes global bijectivity possible. The main targeted application is the creation of correspondence between two objects. Our approach simplifies this by using a combination of a *kd*-tree and a linear array named *uv*-map, which stores tuples of (u,v) and provides fast and efficient two way searches.

The approach is limited by the heavy dependency on the users input. The segmentation process is entirely controlled by the user as is the assignment of regions between two parametrized meshes. Here lies further potential for improvement. A fully automated segmentation method, which leads to reproducible partition and comparable results between different, but similar objects, would enhance the application of this approach. We will investigate current and future segmentation algorithms for suitability.

Our limitation to convex patches is also subject to possible research, as other parametrizations do not depend on convex meshes. By surpassing this limitation it could be possible to completely omit the segmentation and any user input. We will investigate this, too.

ACKNOWLEDGEMENTS

We want to thank the anonymous reviewers for their valuable comments.

5 REFERENCES

[1] Yehuda Afek, Noga Alon, Omer Barad, Eran Hornstein, Naama Barkai, and Ziv Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, Vol. 331, No. 6014:183–185, 2011.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[3] Robert Brown. A brief account of microscopical observations made in the months of june, july and august, 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *Philosophical Magazine*, Vol. 2:161–173, 1828.

[4] Patrick Degener, Jan Meseth, and Reinhard Klein. An adaptable surface parameterization method. In *12th International Meshing Roundtable*, 2003.

[5] Boris N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, 7:793–800, 1934.

[6] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, Vol. 21:209–218, 2002.

[7] Gianni DiCaro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, Vol. 9:317–365, 1998.

[8] Manfredo P. do Carmo. *Differential geometry of curves and surfaces*. Prentice Hall, 1976.

[9] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *ACM SIGGRAPH*, 1995.

[10] Michael S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, Vol. 14, No. 3:231–250, 1997.

[11] Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, Vol. 20, No. 1:19–27, 2003.

[12] Alfred Gierer and Hans Meinhardt. A theory of biological pattern formation. *Kybernetik*, 12:30–39, 1972.

[13] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. In *ACM SIGGRAPH*, 2002.

[14] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In *ACM SIGGRAPH*, 2000.

[15] Kai Hormann and Günther Greiner. *MIPS: An Efficient Global Parameterization Method*. Vanderbilt University Press, 2000.

[16] Liliya Kharevych, Boris Springborn, and Peter Schröder. Discrete conformal mappings via circle patterns. *ACM Transactions on Graphics*, Vol. 25, No. 2:412–438, 2006.

[17] Aaron Lee, Hugues Hoppe, and Henry Moreton. Displaced subdivision surfaces. In *ACM SIGGRAPH*, 2000.

[18] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *ACM SIGGRAPH*, 2002.

[19] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1996.

[20] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Math*, Vol. 2, No. 1:15–36, 1993.

[21] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *ACM SIGGRAPH*, 2001.

[22] Alla Sheffer and Eric de Sturler. Surface parameterization for meshing by triangulation flattening. In *9th International Meshing Round Table Conference*, 2000.

[23] Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. Abf++: Fast and robust angle based flattening. *ACM Transactions on Graphics*, Vol. 24, No. 2:311–330, 2005.

Name	**Distortion minimized**	**Bijektivity**	**Boundary**	**Source**
Uniform	none	yes	convex	[27]
Harmonic	angular	no	convex	[9]
Shape preserving	angular	yes	convex	[10]
Mean-value	angular	yes	convex	[11]
DCP	angular	no	free	[6]
LSCM	angular	no	free	[18]
MIPS	angular	yes	free	[15]
ABF/ABF++	angular	yes (only local)	free	[22]/[23]
Circle patterns	angular	yes (only local)	free	[16]
Stretch minimizing	distance	yes	free	[21]
MDS	distance	no	free	[29]
Adaptable surface	area	yes	free	[4]
Our approach:				
Diffusion-based distance	to a degree: yes (only local)	convex		

Table 1: Comparison of parametrizations with our approach

[24] Alla Sheffer, Emil Praun, and Kenneth Rose. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision*, Vol. 2, No. 2:105–171, 2006.

[25] Alan Mathison Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, Series B, Vol. 237, No. 641:37–72, 1952.

[26] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph.*, 25:289–298, 1991.

[27] William T. Tutte. How to draw a graph. *London Mathematical Society*, Vol. 13:743–768, 1963.

[28] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Computer Graphics*, pages 299–308, 1991.

[29] Gil Zigelman, Ron Kimmel, and Nahum Kiryati. Texture mapping using surface flattening via multi-dimensional scaling. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 8, No. 2:198–207, 2002.

Realistic facial expression synthesis of 3D human face based on real data using multivariate tensor methods

Jacey-Lynn Minoi
Faculty of Computer Science and Information Technology
Universiti Malaysia Sarawak
94300 Kota Samarahan
Sarawak, Malaysia
jacey@fit.unimas.my

Duncan F. Gillies
Department of Computing
Imperial College London
180 Queen's Gate, London,
SW7 2RH, U.K.
d.gillies@imperial.ac.uk

Amelia Jati Robert Jupit
Faculty of Computer Science and Information Technology
Universiti Malaysia Sarawak
94300 Kota Samarahan
Sarawak, Malaysia
rjajati@fit.unimas.my

ABSTRACT

This paper presents a novel approach of facial expression synthesis and animation using real data sets of people acquired by 3D scanners. Three-dimensional faces are generated automatically through an interface provided by the scanners. The acquired raw human face surfaces went through a pre-processing stage using rigid and non-rigid registration methods, and then each of the face surface is synthesized using linear interpolation approaches and multivariate statistical methods. Point-to-point correspondences between face surfaces are required in order to do the reconstruction and synthesis processes. Our experiments focused on dense correspondence, as well as, to use some points or selected landmarks to compute the deformation of facial expressions. The placement of landmarks is based on the Facial Action Coding System (FACS) framework and the movements were analysed according to the motions of the facial features. We have also worked on reconstructing a 3D face surface from a single two-dimensional (2D) face image of a person. After that, we employed tensor-based multivariate statistical methods using geometric 3D face information to reconstruct and animate the different facial expressions.

Keywords

Three-dimensional facial animation, facial expression synthesis, face reconstruction.

1. INTRODUCTION

Facial animation is complex and difficult to achieve realistically. Facial features that contribute the most to facial expressions are the eyelids, eyebrows and mouth. Wrinkles and budges also contribute to the change of facial appearances. Movements or the flow of features can be measured and then used to animate facial expressions. This approach is known as feature-based deformation. [Ste81] used landmark information to deform face shapes and models while [Wat87] used pseudo muscles for face expression animation. Work by [Gue98, Pig98] used facial movement information.

According to [Fas02], the deformation approach does not necessarily require extensive facial movement, which makes the animation process faster and simpler. However, this approach is unreliable in creating exaggerated realistic face shapes and facial expressions.

A large number of facial modelling and facial animation works have employed muscle-based approaches [Ter90, Lee95]. Synthetic facial movements are generated by mimicking the contraction of facial muscles. This can be done by firstly defining the functionality and locations of the facial muscles on the face model and then applying a combination of muscle contractions [Wat87]. The combinations of the muscles are defined by Action Units (AUs) from the Facial Action Coding System (FACS) framework. Using AUs could reduce the amount of work in characterising facial expression data.

Many face animators, [Fox05] for example, imitate facial muscles movements to generate facial expressions. Similar to the prior muscle-based

animation problems, this approach only creates a limited set of facial expressions.

Multi-layer models supplement the use of facial muscles. A multi-layer model is built from the anatomical structure of the face, facial muscles, skin, soft tissues and etc. [Ter90, Lee95, Sif05, Wil97]. [Ter90] proposed facial animation by contracting synthetic facial muscles embedded in a face skin model. This approach improves the realism of the synthetic facial expressions; however the use of sophisticated biomedical models requires accurate simulation methods and high computational costs [Lee95]. Furthermore, animating expressions in complex multi-layer structures requires non-linear methods to simulate dynamic deformation of the skin. Failing to create a detailed skin deformation (such as wrinkles) may result in less realistic facial expressions [Ers08].

The geometry warping approach is another method for synthesizing facial expressions. The facial expression information is measured from two images – one with a neutral expression and another with a particular facial expression. The calculated facial movement difference vectors are transferred to a target image of a neutral face [Pig98, Sha01, Par96, Wil90]. The facial movement differences can be controlled by using linear interpolation. The disadvantage of this approach is that the overall shapes of the face, including the geometric shapes, poses and orientations, and facial expression, are calculated and computed together. It is therefore not a perfect solution for generating the in-between facial expressions [Ers08]. Recent work has been undertaken to overcome this weaknesses by using non-linear interpolation or by combining linear and non-linear interpolation.

[Par96] used simple geometric interpolation to synthesize expression on 3D face models. The feature points are manually digitised on each face model. This was followed by automatic expression synthesis where the data of real actors are captured and analysed [Wil90, Ber85, Ess96]. The captured face surfaces are represented using a structured mesh, along with texture information.

Segmenting face models into smaller regions is also employed with the aim of synthesizing only the relevant parts of the face contributing to an expression. [Jos03] applied this approach on 3D face models. On the other hand, [Bla03] employed a morphable technique to animate facial expressions on existing 2D images and videos. The advantage of the morphable modelling approach is that it can work on faces without acquiring examples of facial expression data of a person. [Vla05] mapped facial movements from a recorded video to a target face using an optical flow-based tracker to estimate 3D shape movements. In addition, they used a multilinear model to manage the face attributes separately. Theoretically, by using multilinear models on a larger collection of faces with different expressions, faces with any expressions can be generated. However, the collection used in Vlasic's work is limited in size. Nevertheless, the advantage of this technique is that visible facial markers or special face-spanning equipment is not required.

In order to simulate a realistic facial expression, a larger collection of facial expression examples is required. When using muscle information, accurate muscle descriptions or templates are needed to produce visually correct facial movements. Using fewer facial expression resources means that expressions may appear artificial and unrealistic.

Methods to optimise the animation computation may also be needed to allow real-time facial animations. It should be noted that facial animation field has grown into a complicated and broad subject. Facial animation applications are extensively used in various areas, including movie industries, computer games, medicine and telecommunication.

The remainder of this work is organised as follows. In Section 2, we start with a description of the data set of 3D face scans from which our synthesis of facial expression model is built.

In Section 3, we briefly present the pre-processing technique used on raw 3D face surfaces, followed by a description on the algorithm for synthesizing and animating facial expressions using linear interpolation based on landmark placements, is given in Section 4.

In Section 5, we describe PCA and LDA approaches used in this work and then introduce our idea of applying tensor model to those two approaches on generating a variety of facial expressions that can be applied in differing degrees. Following that, we look at reconstructing 3D faces from 2D photographic images of faces with only neutral expression and then map facial expression onto the reconstructed face surface.

Section 6 then describes all the experiments that were carried out in the study, and present the results of the synthesis on 3D face data. Finally, in Section 7, we conclude the paper, summarising its main contributions and describing possible future work.

2. FACE DATA SETS USED

In our experiment, we have used four face data sets of real human faces: the Notre Dame 3D face data set, the Imperial College 3D face data set, SUNY Binghampton 3D face data set, and the FERET 2D face data set.

The Notre Dame data set was acquired at the Computer Vision Research Lab at the University of Notre Dame (see web page, http://www.nd.edu/~cvrl/CVRL/Data_Sets.html). A total of 150 subjects participated in the image acquisition sessions, giving a total of 300 three-dimensional face surfaces. The 3D data was captured using a Minolta Vivid camera [Kon] which uses a structured light sensor to scan surfaces. The captured faces are only frontal faces of neutral facial expression.

In the Imperial College face data set was acquired at the Department of Computer Science at Imperial College London [Pap05]. The 3D face surfaces are captured using a VisionRT stereo camera [Vis]. It contains a set of 60 individual face surfaces of which we used two expressions, one where the subject is frowning and another smiling – totalling to 120 three-dimensional face surfaces. Each face is associated with greyscale texture image.

In the SUNY Binghampton data sets [Wan06], the 3D face surfaces consists of 7 different emotional facial expressions, namely anger, disgust, happiness, sadness, surprise, fear and neutral. Each of the emotional expression contains 4-levels of expression intensity ranging from low to high. In total, there are 2,500 faces from 100 subjects.

FERET face data set is a well-known 2D standard face image data set normally used for evaluating face recognition performance. See web page, http://www.itl.nist.gov/iad/humanid/feret/feret_master.html).

3. PRE-PROCESSING

The obtained 3D face surfaces using current technologies would require pre-processing procedure before they can be further analysed. A raw surface may have holes or spikes caused by acquisition and measurement errors. Raw surfaces also have different alignment and surface areas in addition to having a different number of points in the surface representation. The first step in any technique using statistical shape modelling is to normalise the surface with respect to the orientation and the surface area. The second step is then to establish point-to-point correspondence between the surfaces in the input database.

As a result of these steps, extraneous portions of a surface are removed so that all the surfaces cover the same features and are represented by the same number of corresponding vertices. The dense correspondence is established so that, ideally, each point on a surface represents the same anatomical position on all the other surfaces. Thus, each vertex is a landmark point once the correspondence is established. In our experiments, correspondence between 3D face models was established using the method proposed by [Pap05]. Pre-processing is applied to each raw face surface to regularise the position, scale and surface tessellation, and to establish a correspondence between faces.

4. SYNTHESIZING EXPRESSIONS USING LINEAR INTERPOLATION BASED ON LANDMARK PLACEMENTS

In this section, we describe the use of features points that gives parameters of facial expression and deformation. Using these parameters, one can generate any possible facial expressions according to the selected landmark points.

In our experiment, we selected two sets of identifiable landmark points, whereby set 1 has thirty-three landmarks and set 2 has forty-three landmark points. Figure 1(a) illustrates thirty-three landmarks and they are placed along the eyebrows, the corners of the eyes and crowfeet, the glabella, the upper part and the tip of the nose, the mouth and areas around the mouth and lips, the chin and the cheeks. Figure 1(b) shows set 2 landmark points. The chosen landmark points are based on the landmarks used in craniofacial anthropometry and muscle-based landmarks in the FACS framework.

The main source that contributes to expression variation is facial muscular movements. We employed a registration framework [Pap05], based on the selected facial landmarks, to create expressions on surface model of the face.

The step is similar to the geometric warping approach, whereby we calculate the facial movement difference vectors from two face surfaces – one face with a neutral expression and another with a particular facial expression – using landmark points. Then, we use linear interpolation to transfer the generated facial expression to a neutral face.

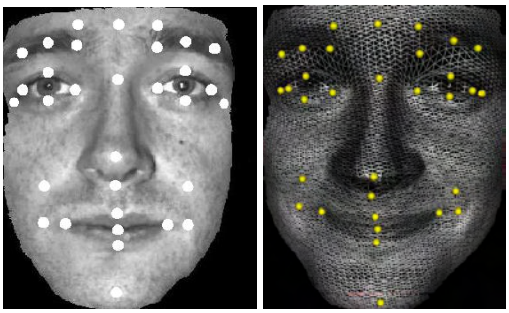


Figure 1(a). A set of 33 selected points.

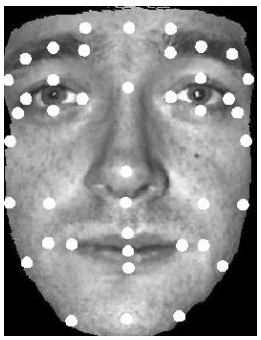


Figure 1(b). A set of 43 selected points.

5. SYNTHESIZING EXPRESSIONS USING MULTIVARIATE STATISTICAL METHODS

In this section, we present another approach to generate and synthesize facial expressions using multivariate statistical method based on PCA and mLDA (Maximum uncertainty Linear Discriminant Analysis). And then we introduce a tensor-based multivariate statistical method to construct new face shapes with a range of different face variations.

Active Shape Model (ASM) [Coo95] is a commonly used approach to build statistical shape models of the human face. The modelling of anatomical face structures are from statistical information found in a training set. Unlike the multivariate statistical method and tensor-based multivariate statistical approaches, new face shapes are created based on statistical information from pre-defined classes of specific features or face variations found in the training set.

Our idea of using tensor model on the multivariate statistical method is to use all the face features with a variety of facial variants simultaneously rather than separating them into two numbers of classes. The advantage with this method is that it is practical to generate a variety of face shapes applied in different degrees. Additionally, the transition between face shapes is also continuous and natural.

Following that, we applied the tensor-based method to the reconstructed 3D faces from 2D photographic images and synthesize facial expressions.

Revisiting PCA and LDA

Principal Component Analysis (PCA) is one of the most successful methods to reduce the dimensionality of the original space with a minimum loss of information by finding the projection directions that maximise the total scatters across all data. However, in the covariance structure of PCA, the first principal component with the largest eigenvalue does not necessarily represent the important discriminant directions to separate sample groups. Therefore, we employ the idea of using the discriminant weights given by separating hyperplanes to select among the principal components the most discriminant ones. Linear Discriminant Analysis (LDA) is computed to separate samples of distinct groups by maximising the ratio of the determinant of the between-class separability to the determinant of the within-class variability. The performance will degrade if there are only a limited number of total training samples N compared to the dimension of the feature space n. This critical issue is the singularity and instability of the within-class scatter matrix. In order to avoid these problems, we propose to use Maximum uncertainty Linear Discriminant Analysis (mLDA) approach. The idea of mLDA is to regularise the eigenvalues. The details of the mLDA method can be found in [Tho06].

Multivariate statistical method

The multivariate statistical method is essentially a two-stage approach, the first stage is to characterise a type of variation and the second stage is to reconstruct faces. The method is used to find the most significant direction of change between two classes, and to reconstruct and visualize intermediate data between two classes. This method is based on Principal Component Analysis (PCA) and Maximum uncertainly Linear Discriminant Analysis (mLDA) separating hyper-plane. This technique was first applied by [Kit06] to extract and characterise the most discriminant changes between two groups of 2D probed images.

The initial training set of 3D face data consisting of N training examples on n variables is managed by dividing the training data into two groups or classes, C_1 and C_2. The training datasets can be projected from the original vector space of N by n to a lower dimensional space using a full rank PCA transformation. The principal component space forms an $n \times m$ transformation matrix, where $m = N - 1$. This step may or may not be necessary to overcome the singularity of the within class covariance matrix. If $N \geq n$, then PCA transformation is not required. It is possible that after PCA dimensionality reduction, the within-class scatter matrix S_w may still be less than full rank. If so, the mLDA approach is used to ensure that the scatter matrix S_w is non-singular. Ordering the eigenvectors is not necessary for this process. As there are only two classes, $g = 2$ and the resulting mLDA is a unidimensional vector of length m (the linear discriminant eigenvector has dimension $m \times 1$).

Back projecting into the original data space, the most discriminant feature is a n x 1 vector. The final step in the first stage is to calculate the mean of each group and the corresponding variances on the unidimensional space. This is a very fast computation because we are dealing with one-dimensional data from two-group classification problem. To reconstruct these discriminant points (based on standard deviations and means) on the original space, we do the inverse steps. Once the classifier has been constructed, we then extract and project the discriminant vector. This can be done simply by converting the discriminant to its corresponding n x 1 dimensional face vector.

The final stage of the PCA+mLDA method is the reconstruction step. If we project the most discriminant vector found for the two classes into the original data space, we will obtain a n x 1 vector. Moving a point in the original data space in this direction will change the point from an example of one class to a maximum likelihood estimate of that point in the other class. Assuming that the spread of each class follows a Gaussian distribution, the limits of variation can be set to $\pm 3sd_i$, where sd_i is the standard deviation of each class i. By moving along the n x 1 dimensional most discriminant features based on the mean of each group and the corresponding standard deviations of each group, the face shapes according to the class variant can be reconstructed in the original face domain. As stated before, since there is only one dimensional data from a two group classification problem, the computation is very fast. Figure 2 shows the geometric overview of the two-class multivariate statistical approach. This method is then extended by using tensor model to 3D face shapes to allow multiple numbers of classes.

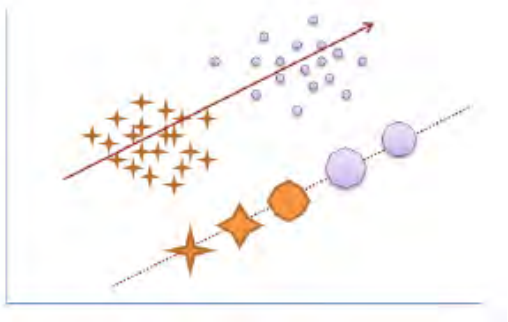


Figure 2. The geometric overview of the multivariate method between two classes.

Tensor-based multivariate statistical method

A tensor is a multidimensional matrix or mode-n matrix and is useful for the description of higher order quantities. The N^{th} order tensor is written as $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, where the I_N represents the mode-n space. A tensor is then flattened (see [Lat00] for flattening details) into a matrix form, A_n, along any dimension n where $n = I_1, I_2, \ldots, I_N$.

Starting with a dataset of 3D face surfaces, we organise the data in a tensor model according to N varieties of face shapes. In our experiment, the training set is arranged into a tensor explicitly accounts for facial expression variation, where the core tensor manages the interaction between the indices of the 9-mode matrices, ($I_{subject}$ x I_{anger} x $I_{disgust}$ x I_{fear} x I_{happy} x I_{sad} x $I_{surprise}$ x $I_{neutral}$ x I_{points}). Next, we perform matrix unfolding to retrieve a set of the basis matrices for all the 9-mode matrices. Then, we compute the left singular value matrices using Singular Value Decomposition (SVD) method to obtain U_N matrices. Each of the U_N matrix can be thought of as the principal components in each mode, and they may not necessarily be of the same dimension as the tensor. The generalised N-mode SVD can be written as follows, and can be interpreted as a standard linear decomposition of the data ensemble.

$$A_n = U_n \bullet D_n (U^{n-1} \circ \ldots \circ U^1 \circ U^N \circ \ldots \circ U^{n+2} \circ U^{n+1})^T$$

The '$\circ$' is denoted as Kronecker product [Rao71] and it is applied to compute the product of the matrices. All the computed I sets of non-zero eigenvalues are extracted and stored differently according to the features of interest and are not ordered.

Since we have I-group classification, there are $g_i = g_{i-1} + (i - 1)$ number of discriminant vectors, where $g_1 = 0, i \in \{1, 2, \ldots I\}$ is the number of group classification based on the feature of interest and each represent the most expressive features. The resulting mLDA now has multiple coefficient vectors in a dimension of $\{g \times (m \times 1)\}$ depending on the choice of facial expression transformations. Next, is to compute the second stage of multivariate statistical method. Based on the feature of interest, we then determine the most discriminant vectors that best characterise the particular change in facial features.

Tensor-based multivariate statistical method is used on the SUNY Binghampton 3D face dataset and the reconstructed 3D faces from 2D images of neutral faces.

3D Face Reconstruction

The reconstruction method has four distinct steps, which are: (1) 3D-2D alignment, (2) texture mapping,

(3) illumination adjustment, and (4) shape estimation. The detailed methods can be found in [Has07]. First, the 2D image, which is to be reconstructed, is landmarked manually by hand using a set of uniquely identifiable points with the same landmarks already known in the 3D statistical shape model. This landmarking is used to establish correspondence between the 2D image and the 3D model, which then allows mapping of the 2D image texture to the 3D face surface. The objective of the shape estimation is to optimize the match between the projection of the 3D shape model and the original 2D image. This is carried out by adjusting the shape parameters with the most discriminant vector. Figure 3 illustrates the 2D-to-3D reconstruction approach.

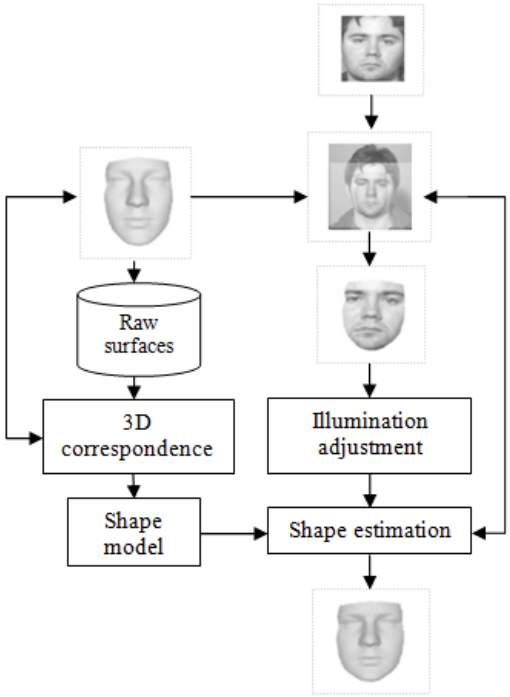


Figure 3. The overall process of 3D face reconstruction (adapted from [Has07]).

Here, we look at reconstructing 3D faces from 2D photographic images of faces with no facial expression and then generating realistic human facial expressions. In this case, frontal images from FERET and Notre Dame Face databases are used for evaluating facial expression synthesis on unseen subjects.

6. EXPERIMENTS AND RESULTS

Three facial expression synthesizing and animation experiments with three different face data sets were performed.

In the first experiment, we selected two sets of identifiable landmark points and then employ linear interpolation method. Figure 4 compares the results of the synthesis of a smile on 3D human faces using 33 and 43 landmark points. The results show an obvious geometric distortion to the overall shapes of the face (see the distortion on the nose at the bottom row of the figure), and using only 33 landmarks do not reconstructed a noticeable smile.

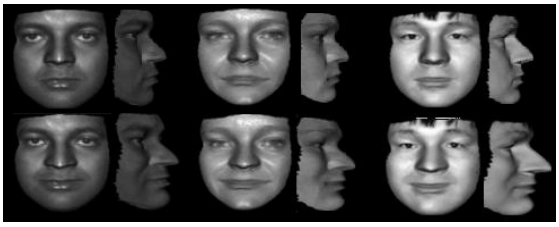


Figure 4. Synthesis of a smile using 33 landmarks (top row) and 43 landmarks (bottom row).

Figure 5 shows an example of an unrealistic face shape with a reconstructed smile using higher number of landmark points, which are placed mainly around the cheek. This shows that the number and the placement of landmark points on a face are critical to produce a realistic facial expression.

The disadvantage of using the linear interpolation approach is that the overall geometric shape of the face and the facial expression are computed together. Furthermore, there is no way to generate the in-between facial expression to allow a smooth synthesis and animation of facial expressions.

Figure 5. (a) Original smile; (b) Reconstructed caricature smile.

In the second experiment, we employed tensor-based multivariate statistical method to the SUNY Binghampton face data set. We compared the output of the synthesis with ASM.

The results of the reconstructed facial expressions using the most expressive features captured using ASM is as illustrated in Figure 6. The reconstructed

faces are restricted by limiting the change in each principal component to $\pm 3\sqrt{\{\lambda_i\}}$, where λ_i are the corresponding largest eigenvalues. The first mode describes the vertical stretch along the centre of the face. The second mode models the variations in the horizontal direction. The third mode captures variation around the mouth and cheek to create expression changes from distorted frowning expression to a caricature smile.

Figure 7 illustrates the facial expression transformations of the first four largest principal components captured by ASM on the Imperial College data set. The first mode describes the horizontal stretch around the cheek and mouth. The second mode models the variation in the vertical direction. The third mode captures variations around the nose and eyes areas. The fourth mode captures the horizontal variation of the geometric shape of the face.

By examining Figure 6 and Figure 7, we see that the face shapes are not properly grouped according to facial expressions. The changes of face shape are global to the data set which makes it impossible to synthesize individual facial expression. Thus, ASM method is not suitable to capture specific facial expression variations.

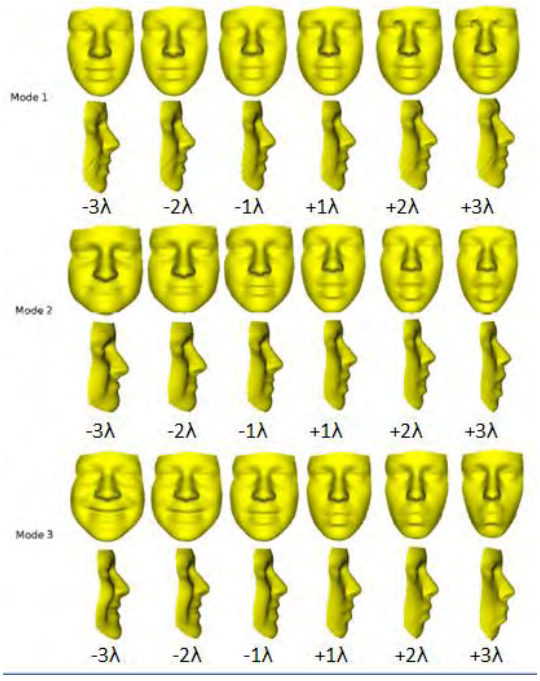


Figure 6. Synthesis of facial expressions reconstruction using the most expressive principal components captured by ASM.

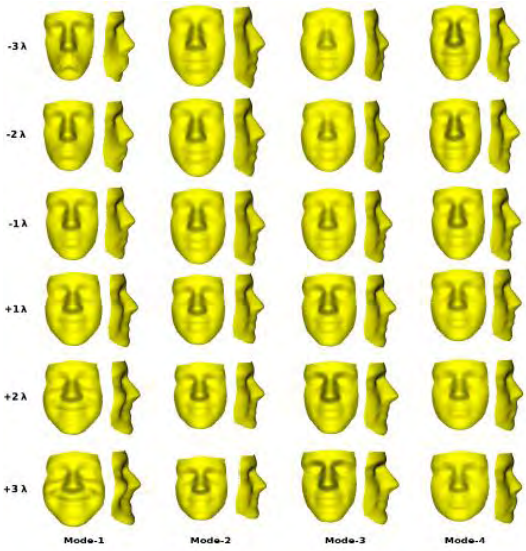


Figure 7. Synthesis of facial expressions reconstruction using the most expressive principal components captured by ASM using Imperial College face data set.

Figure 8 shows the results of the reconstruction for a neutral and an angry face expression using the tensor-based multivariate statistical method. Figure 9 illustrates the synthesis of facial expressions between a surprise and a frowning face.

This method is effective to capture facial expressions variation and it is able to find the most characteristic direction of change involved in an expression. This magnitude of change can be controlled by a single scalar magnitude. We explore the reconstruction and synthesis of face shapes by moving the point from one side of the dividing hyper-plane to the other, respecting the limits of the standard deviation and the measured mean of each sample group.

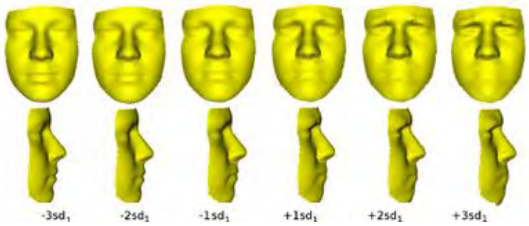


Figure 8. Synthesis from a neutral to an angry expression using most characteristic direction captured by tensor-based multivariate statistical method.

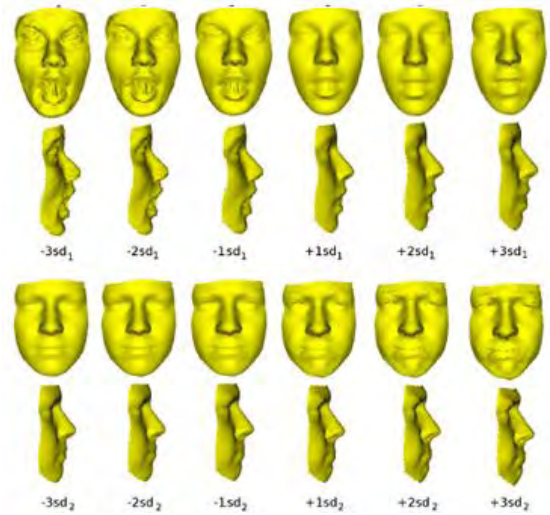


Figure 9. Synthesis of a surprise to a frowning expression using the most characteristic direction captured by tensor-based multivariate statistical method.

In the third experiment, we work on synthesizing realistic facial expressions on a reconstructed 3D real human face given only a single frontal 2D face image. We tested the technique using the 2D FERET face data set. Figure 10 shows the reconstructed 3D face shapes from faces taken from FERET face images and the synthesized facial expressions to the reconstructed 3D face shapes. The fourth column from the left of the figure displays the original faces. As we move from the original to the left side of the figure, a range of smiling expressions is generated. Similarly, when we move to the right side of the figure, a range of frowning expressions is generated. Having texture embedded to the 3D face surfaces makes the expression change smoother and more visible. For example, the raised cheeks and eyebrows, and the opened mouth show a smile.

Figure 11 shows the synthesis and animation of facial expressions on the Notre Dame 3D face data set, given that Notre Dame 3D faces only contain neutral face (the fourth column from the left of the figure).

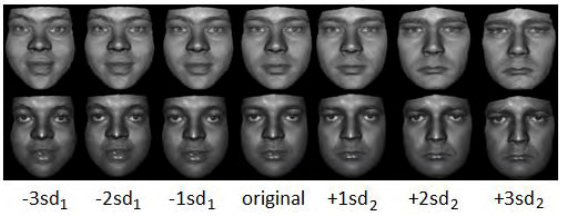


Figure 10. The reconstruction of smiling and frowning expressions using tensor-based multivariate statistical approach.

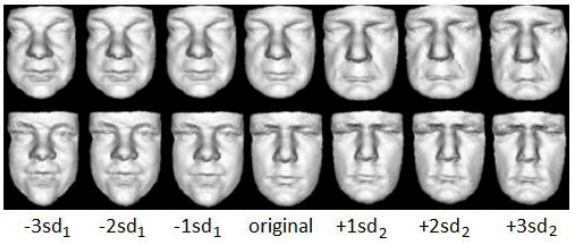


Figure 11. The reconstruction and synthesis of the most characteristic component when using 3D Notre Dame faces along the smile and frown expressions.

Examining Figures 8 to 11, we can clearly see that the tensor-based multivariate statistical approach effectively extract the 3D facial expression changes. In fact, this approach is also able to generate a gradual change on facial expressions that is not explicitly present in the training data sets.

7. CONCLUSIONS

This paper describes 3D facial expression animation using real human faces. We have analysed the placement of landmarks based on FACS for deformation and synthesis of facial expressions. Unfortunately, landmark-dependent may not create realistic facial expressions. We introduce another method, which is the multivariate statistical method, which differs from many other synthesizing and facial expression animation approaches in terms of using the whole face data points instead of selecting feature points or landmarks on the face for shape variations.

This approach could extract facial expression characteristic discriminant information efficiently, providing a gradual transformation on the 3D faces. The strength of this work is the realism of the facial expression generated as we use and extract only facial expressions from real human faces. We could also generate facial expressions at varying intensities for a subject without prior examples of expression.

The concept of using PCA+ mLDA approach to discriminate pattern of interest is not new. However, in the work of real human 3D faces, synthesizing and analysing facial expression is still in its preliminary stage. We have also implemented to using tensor model to extend the two-class problem to several classes.

8. ACKNOWLEDGMENTS

The authors would like to thank Prof. Carlos Thomaz for his technical advices, the referees for their constructive comments which helped to improve this paper, and Universiti Malaysia Sarawak (UNIMAS) for the travelling fund.

9. REFERENCES

[Ste81] Stephen M. Platt and Norman I. Badler. "Animating facial expressions." SIGGRAPH Computer Graphics, 15(3):245–252, 1981.

[Wat87] K. Waters. "A muscle model for animation three-dimensional facial expression." International Conference on Computer Graphics and Interactive Techniques, pages 17–24, 1987.

[Gue98] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin. "Making faces." Annual Conference on Computer Graphics and Interactive Techniques, pages 55–66, 1998.

[Pig98] F. Pighin, J. Hecker, D. Lischinski, and R. Szeliski. "Synthesizing realistic facial expressions from photographs". In SIGGRAPH Computer graphics, pages 75–84, Orlando; FL, 1998. ACM Press.

[Fas02] B. Fasel and J. Luettin. "Automatic facial expression analysis: a survey". IDIAP Research Report 99-19, 2002.

[Ter90] D. Terzopoulos and K.Waters. "Physically-based facial modeling, analysis and animation". Journal on Visualisation and Computer Animation, 1(2):73–80, 1990.

[Lee95] Y. Lee, D. Terzopoulos, and K. Walters. "Realistic modeling for facial animation". In SIGGRAPH on Computer Graphics and Interactive Techniques, pages 55–62, New York, NY, USA, 1995.

[Fox05] B. Fox. "Barrett fox character animator". Internet: http://www.barrettfox.com/, 2005.

[Sif05] E. Sifakis, I. Neverov, and R. Fedkiw. "Automatic determination of facial muscle activations from sparse motion capture marker data." ACM Transactions Graphics, 24(3), 2005.

[Wil97] J. Wilhelms and A.V. Gelder. "Anatomically based modeling." Annual Conference on Computer Graphics and Interactive Techniques, 1997.

[Ers08] N. Ersotelos and F. Dong. "Building highly realistic facial modeling and animation: a survey." The Visual Computer, 24(1):13–30, 2008.

[Sha01] A. Shashua and T. Riklin-Raviv. "The quotient image: class-based re-rendering and recognition with varying illuminations." IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(2):129–139, 2001.

[Par96] F.I. Parke and K. Waters. "Computer facial animation". A K Peters, 1996.

[Wil90] L. Williams. "Performance-driven facial animation." Annual Conference on Computer Graphics and Interactive Techniques, pages 235–242, 1990.

[Ber85] P. Bergeron and P. Lachapelle. "Controlling facial expressions and body movements in the computer-generated animated short "tony de peltrie"." In ACM SIGGRAPH Advanced Computer Animation seminar notes, 1985.

[Ess96] I. Essa, S. Basu, T. Darrell, and A. Pentland. "Modeling, tracking and interactive animation of faces and heads using input from video." IEEE International Conference on Computer Animation, 00:68, 1996.

[Jos03] P. Joshi, W.C. Tien, and M. Desbrun. "Learning controls for blend shape based realistic facial animation." ACM SIGGRAPH Eurographics Symposium on Computer Animation, pages 187–192, 2003.

[Bla03] V. Blanz, C. Basso, T. Poggio, and T. Vetter. "Reanimating faces in images and video." Annual Conference of the European Association for Computer Graphics, 22(3):641–650, 2003.

[Vla05] D. Vlasic, M. Brand, H. Pfister, and J. Popovic. "Face transfer with multilinear models." ACM Transactions on Graphics, 24(3), 2005.

[Kon] Konica Minolta, Vivid 910. [Online].
Available: http://www.minoltausa.com

[Pap05] T. Papatheodorou and Daniel Rueckert. "Evaluation of 3d face recognition using registration and PCA." In Takeo Kanade, Anil K. Jain, and Nalini K. Ratha, editors, AVBPA, volume 3546 of Lecture Notes in Computer Science, pages 997-1009. Springer, 2005.

[Vis] VisionRT. [Online]. Available: http://www.visionrt.com

[Wan06] J. Wang and L. Yin and X. Wei and Y. Sun. "Facial expression recognition based on primitive surface feature distribution." in the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR 2006), New York, 2006.

[Coo95] Cootes, T., Taylor, C.J., Cooper, D.H., Graham, J. "Active shape models - their training and application." Computer Vision and Image Understanding, 61(1): 3859. 1995.

[Tho06] C.E. Thomaz, E.C. Kitani and D.F. Gilles: "A Maximum uncertainty LDA-based approach for limited size problems with applications to face recognition." Journal of the Brazilian Computer Society, Vol 12(2), pg 7-18, 2006.

[Kit06] Kitani, E.C., Thomaz C.E., Gillies, D.F. "A Statistical discriminant model for face interpretation and reconstruction." In: 19th Brazillian Symposium on Computer Graphics and Image Processing. 2006.

[Lat00] L.D. Lathauwer, B.D. Moor, and J. Vandewalle. "A multilinear singular value decomposition". SIAM Journal on Matrix Analysis and Applications, 21(4):1253–1278, 2000.

[Rao71] C.R. Rao and S. Mitra. "Generalised inverse of matrices and its applications". Wiley New York, 1971.

[Has07] S. Hassan Amin and Duncan Gillies. "Quantitative analysis of 3d face reconstruction using annealing based approach". In: IEEE International Conference on Biometrics: Theory, Applications, and Systems, BTAS 2007, pg 1–6, 2007.

Procedural generation of meandering rivers inspired by erosion

Michał Kurowski

Warsaw University of Technology, the Faculty of Electronics and Information Technology
Pl. Politechniki 1
00-661, Warsaw, Poland

M.Kurowski@ii.pw.edu.pl

ABSTRACT

This paper describes a method of procedural generation of meandering rivers inspired by erosion, which can enhance visual realism of virtual terrains. Terrain is represented using an adaptively subdivided triangle mesh with additional information (e.g. amount of soft deposit) stored in vertices. Water is simulated using Smoothed Particle Hydrodynamics (SPH), modified in order to model erosion occurring within meanders. Most experiments were performed on an initially flat terrain, so in order to provide the initial disruption of an otherwise straight flow, a simple force simulating an exaggerated Coriolis effect was introduced.

Keywords

Computer graphics, procedural terrain generation, meanders, Smoothed Particle Hydrodynamics, erosion.

1. INTRODUCTION

Creation of "virtual worlds" used in computer games, simulations, movies or art requires a significant amount of various content, including, but not limited to, textures, object models, sounds and terrains. This content can be either handcrafted by skilled professionals or generated procedurally. This distinction is not very sharp, as various packages offer the ability to combine both approaches (e.g. map editor in "Earth 2150" game), by allowing their users to procedurally generate some elements of the content (e.g. fractal terrain for use in a Real-Time Strategy game) and then manually combine and tweak them to match specific requirements (in case of the previous example: create a level patch of terrain for player's base or place resources and bridges). The growing computational power of devices in the hands of end-users is followed by their growing expectations for visual quality, which is often linked to visual complexity [Mus02]. This in turn leads to the growing amount of required work, which translates to development time, staff size and budget size, making the procedural approaches more noteworthy. Automatic generation is already very attractive to indie game developers who are aiming for low cost and low development time instead of high complexity within an acceptable budget.

Terrain is present in many "virtual worlds" and often strongly influences the final product: beautiful landscapes create atmosphere in movies and complement the action, while maps for strategy games decide the gameplay style. A complete and practical solution should be able to produce a visually complex scene in an acceptable time. This scene should contain features desired by an artist or a level designer. Unfortunately, pure procedural approaches are often either hard to control or produce results which look artificially. Methods based on physical simulations are often more intuitive and easy to integrate with other solutions, but they are also either very slow or too simplified to produce certain phenomena, such as for example meanders.

This paper describes a method that can be used to enhance an existing terrain model (created procedurally or manually) with meandering rivers, thus introducing some amount of physically inspired realism. The presented heuristic solution produces meanders by eroding the outer river banks horizontally and depositing the eroded material mainly near the inner banks. It uses SPH for water simulation and an adaptive triangle mesh for terrain representation. The most important contributions of this work are: the introduction of separate material particles, simulating an exaggerated Coriolis effect to initiate the meandering and using a low number of particles to represent water in order to achieve acceptable performance. The resulting method can be used interactively by an artist or a level designer, can

take into account an existing terrain topography and can be further enhanced to allow for different material properties.

2. RELATED WORK

Due to the practical applications of procedural terrain generation many different methods were proposed. In [Mus02] Musgrave covers terrain synthesis using multifractals. Various erosion based approaches were presented in [Nei05], [Sta08] and [Kel88]. [Ben06a] by Benes is a short paper focused on hydraulic erosion introducing some problems that need to be addressed: scene size, simulation speed, user input, interactivity and covering of different scales. Some of these issues are addressed by a GPU based method introduced in [Val11], which uses an adaptively tiled "virtual layered terrain" together with pipe model for water simulation and allows interactive editing. The authors of [Cen09] present automatic terrain generation based directly on user's sketches, while Zhou, Sun and Turk propose a method which combines a "feature map" sketched by the user with example data from a digital elevation model [Zho07]. A similar idea, where a user specifies terrain primitives which are then matched against a database of "terrain units" manually extracted from real-world elevation data is presented in [Chi05]. Brosz [Bro06] proposes a method of enhancing user created model with high-resolution details extracted from another terrain. A fairly recent general overview of procedural techniques for creating 3D environments can be found in "A survey of Procedural Methods for Terrain Modelling" [Sme09].

The problem of meandering rivers is rarely addressed. One of the exceptions is work by Prusinkiewicz and Hammel [Pru93], where a river modeled using a squig curve is incorporated into a mountainous terrain created using midpoint-displacement algorithm. The process involves recursive subdivision of the triangles within the terrain's mesh and classification of the triangles' edges into river entry, river exit and neutral ones. The proposed solution produces complex fractal terrains, but lacks some realism: the river flows on constant altitude in an asymmetric valley. The authors also mention certain issues with tributaries which are not solved in the article.

Belhadj [Bel05] proposed an enhanced midpoint-displacement algorithm which is constrained by a pre-computed set of ridge lines and a river network. The network is created by tracing and combining the trajectories of randomly seeded "river particles". The results look promising and contain meander-like river paths, but their origin is not presented in the paper.

Teoh [Teo08] presented a different approach in WaterWorld. Rivers can be "seeded" by the user and then grown downward cell-by-cell in the direction of the lowest neighbor until they find another body of water. If a section of the river is found to be gently sloping, meanders are generated as an alternating curve defined by "wavelength", which is proportional to the flow of the river and "meander angle" set by the user.

In a more recent work [Teo09] the same author proposed another approach in which ridge lines are created by the user, while the river network is grown inward into the land mass from randomly placed river mouths. Each river is generated by adding consequent, randomly rotated segments and then fitting an alternating curve through them. In order to achieve varied terrain, different rivers have different "SegmentLength" and "MeanderCurvature" parameters. The heightmap is then created to accommodate the generated ridges and rivers.

The authors of [Ben06b] proposed a method based on the Navier-Stokes equations and a regular voxel grid. In one of their experiments they simulated a riverbed with meanders flooded by a wave. The simulation produced excellent results, with the river eroding the outer banks, breaking through the meanders and leaving two billabongs. However, the algorithm has high computational cost and is unsuitable for simulating large terrains (the experiment was conducted on a 120x32x120 grid). Also, the initial meanders were not created during the simulation.

A recent work on hydraulic erosion using CUDA for acceleration [Bez10], while not dealing with meanders, has some similarities with the method presented in this paper. Both use particles for water simulation and adaptive triangle meshes for terrain. However, the solution proposed by Bezin performs the subdivision during an off-line pre-processing and constrains the movement of vertices to the vertical axis (the authors use the term "adaptive heightfield").

3. PROCEDURAL MEANDERS

Proposed Mechanism

Flow of water within a meandering channel is a complex 3D phenomenon which can be resolved into a primary downstream and a secondary transverse components [Nal97]. Conducted experiments [Ram99] show that these components change with depth, distance from banks and along the length of the meander. An exact physically correct simulation (like [Ben06b]) on a large scale would require a substantial amount of computations. On the other hand, a simple curve fitting algorithms (like in [Teo08]) do not take into account local terrain variations. This paper proposes a simple erosion inspired heuristic algorithm, which is less complex than an exact simulations, but can also accommodate these variations and produces plausible results.

The presented method takes into account both erosion and deposition. The "erosion force" f_e is

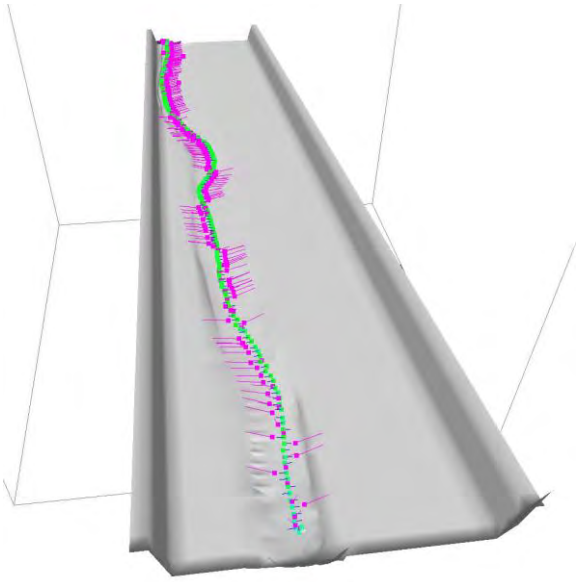


Figure 1. Attraction of the sediment to the inner banks of curves within the river channel. Green particles – water; violet particles (with vectors) – sediment.

proportional to the speed of water v multiplied by its amount w and a "directional factor", which is interpolated from 0 at the inner bank of the meander to 1 at the outer bank. The amount of material that can be potentially eroded m_p depends on the f_e, the area a for which f_e is calculated, the material's softness s and a user specified multiplier u_1:

$$m_p = f_e \, s \, u_1 \, a$$

The actual amount of eroded material m_e is m_p clamped so that water is not oversaturated with sediment. If it weren't for this condition, the multiplication and the consequent division by a wouldn't be necessary. Direction h in which eroded terrain is displaced consists of two components: downward h_y, which is constant, and horizontal $\begin{bmatrix} h_x \\ h_z \end{bmatrix}$, pointing at the outer bank. The magnitude of $\begin{bmatrix} h_x \\ h_z \end{bmatrix}$ is equal 0 where water flows straight and grows to a user defined value when the flow is curved. The terrain's displacement t is calculated as follows:

$$t = \frac{m_e}{a} \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix}$$

The "directional factor" ensures that the outer bank is eroded more than the inner one and $\begin{bmatrix} h_x \\ h_z \end{bmatrix}$ introduces lateral erosion, thus ensuring growth of the meander. It should be noted, that the erosion occurs only when water speed v is greater than a certain user-defined amount. This measure was introduced during early experiments in order to avoid issues with self-deepening or oscillating bed, which occurred in places where water was semi-stationary. However, further research into the mechanics of erosion revealed the existence of "critical shear stress". The introduced threshold is applied to speed, not the actual shear stress, but as both parameters are correlated, it serves the same purpose.

The emergence of meanders is the result of both the horizontal erosion of the outer bank and the deposition of sediment near the inner bank. In order to facilitate the latter process, the eroded material carried by water is attracted toward the inside of the meander (Figure 1). The strength of this attraction is user defined.

Both erosion and deposition create a positive feedback loop which causes the growth of meanders, but also requires some initial curvature. This curvature can be supplied by the shape of the terrain or by an external force. In one of his speeches, Einstein [Ein26] mentioned the role of Coriolis force in the non-uniform distribution of velocities within a flowing river. The magnitude of this force is dependent on the Earth's rotation speed, the current's velocity and the geographical latitude. In order to reduce the amount of user-controlled parameters, the presented method assumes that the changes of latitude are negligible within the simulation's domain. This leads to a simplified formula, which is intuitively consistent with Baer's law and produces satisfactory results:

$$f_c = u \begin{bmatrix} 0 \\ 0 \\ v_x \end{bmatrix} m \quad ,$$

where f_c is the force, u is a user defined factor, v_x is the velocity of water projected onto the X axis and m is the mass of water to which the force is applied.

Water Simulation

The presented method requires the water simulation to provide not only the speed and mass of water in certain areas, but also information whether the stream is curving and in what direction. The process of meandering depends on small, smooth variations in the direction of river flow, which should not be dampened in time. For performance reasons, the simulation should be conducted only in areas containing water, which usually occupy a fraction of the entire domain.

Existing water simulation methods can be divided into Eulerian (operating on meshes or fixed grids) and Lagrangian (operating on particles). Although hybrid methods exist and are used in computer graphics, erosion simulation usually employs pure approaches ([Ben06b] and [Ben09]). The Eulerian methods usually produce excellent, physically correct

results, but are computationally expensive, especially when applied to large 3D domains. They also reveal a tendency to dampen movement which isn't aligned with the used grid. There are works which solve some of these problems by introducing non-regular adaptive meshes or additional particles, but they are usually complicated. Interaction with solid boundaries is either complex or requires high resolution. On the other hand, even simple Lagrangian methods do not suffer from anisotropy and can be easily optimized for large, sparse domains. Current consumer hardware is capable of simulating tens of thousands of particles at interactive rates using GPGPU [Gos10]. Physical accuracy of the simplified methods doesn't match that of the Eulerian ones, but it is usually good enough for applications in computer graphics.

The presented paper uses Smoothed Particle Hydrodynamics, which was introduced in [Mul03] and combines relative simplicity with great flexibility. The characteristic feature of this method is that certain quantities (e.g. pressure) defined at discrete particle locations can be evaluated also in their neighborhoods as continuous values. This is achieved by accumulating contributions from individual particles weighted by radial symmetrical smoothing kernels. The SPH particles used in this paper are enhanced with additional quantities for use only in the erosion algorithm – scalar "curve accumulator" c_a and vector "curve direction" c_d. $c_a$is defined initially as 0 for each particle. If a particle's horizontal velocity vector changes its direction more than a certain value, c_a is increased or decreased depending on whether the velocity was changed to the left or to the right, while c_d is set to left. If the absolute value of c_a is greater than a certain threshold, the stream is assumed to be curving in the direction defined by $c_a\ c_d$. This value is then used as $\begin{bmatrix} \mathrm{h_x} \\ \mathrm{h_z} \end{bmatrix}$. The performance of SPH method relies on fast finding of neighboring particles. A simple regular 2D grid containing indices of particles within a certain volume is used for this purpose.

Sedimentation

Eroded soil is carried with water as sediment. In other works using SPH this information is usually embedded in water particles. The sediment flow and concentration is influenced not only by the water velocity, but also for example by gravity and diffusion. The authors of [Ben09] solve this problem using a donor-acceptor scheme, where the movement of imaginary sediment particles is simulated by material transfer between water particles. This approach produces realistic results, but depends on a large amount of particles and allows the sediment concentration to be defined only in and between them. The method described in this paper strives for river representation using the lowest possible amount of particles and requires the sediment to be attracted towards the inner banks, so that the center of its concentration may not overlap with the center of water mass. For this reason the presented solution uses separate sediment particles that are influenced by the flow of water, gravity and attraction to the insides of the meanders.

Each particle's speed v is calculated as:

$$v_{n+1} = v_n + (g + r + a(s - v_n) + normalize(\begin{bmatrix} \mathrm{h_x} \\ 0 \\ \mathrm{h_z} \end{bmatrix}) fs)t,$$

Where v_{n+1} is the particle's speed in the next simulation step, v_n is the particle's speed in the current simulation step, g is the gravity vector, r is the strength of repulsion from the terrain, a is a user defined "acceleration factor", f is a user specified attraction factor, s is the average speed of the surrounding SPH particles and t is the time delta between the simulation steps.

If new sediment is added to the simulation, the nearest sediment particle is searched for. If such particle is not found, then a new one is created. If the search is successful, a certain amount of the sediment is added to the existing particle, so that the capacity of the particle is not exceeded. The amount that could not be added is returned to the erosion algorithm and is used to clamp the m_p value introduced earlier. This stops erosion when water is oversaturated with eroded material. However, the oversaturation can occur if multiple particles concentrate in a small volume (there are no collisions between them, so they can be arbitrarily close to each other). In this case the sedimentation process is initiated to get rid of the excess material. The sedimentation also occurs as the particle ages, when it loses contact with water (which rarely happens) or its speed is lower than a threshold v_t, calculated based on user-controlled parameter u_2 and the distance from the center of water mass c_w:

$$v_t = u_2\left(1 + \frac{4}{r}\|p - c_w\|\right),$$

where p is the particle's position and r is the smoothing radius of the SPH simulation. This formula is purely heuristic and was developed by experimentation. The situation in which the sediment particle leaves water is undesired, so it's movement is restricted to the volume within $0.99r$ of the nearest SPH particle. This solution works most of the time, but breaks if water velocity changes rapidly. However, these rare stray sediment particles do not seem to introduce any problems to the simulation. For performance reasons, if two particles are close enough and do not exceed the saturation limit, they

are merged together into a larger one. The search for neighbors in a given area is accelerated by a 2D grid identical to the one used in water simulation.

Terrain Representation

The author's first experiments with meandering were conducted on a commonly used heightmap (Figure 3). The directional feature of the erosion was simulated by eroding two points at the same time – the one where the erosion parameters were calculated and its neighbor in the direction of the erosion. The meanders started to form, but their evolution ended fast. When the elevation difference between the river bed and its bank increased, the simplified lateral erosion was less effective, while the sedimentation performance was constant. This lead to silting and overflowing of the bed. A better terrain representation was required.

Further experiments were conducted using "deformable voxels" introduced by the author in [Kur11a]. The initial results were encouraging, but as it turned out, it was difficult to maintain coherence of the grid. An attempt to refine the terrain representation was made. However, before a satisfactory solution was found, rising computational complexity made the idea impractical and without much room for optimization.

Because of the difficulties with voxel representations, a polygonal mesh was used. The collision surface is defined by triangles, while the properties of the terrain are stored within vertices, which are displaced by erosion and sedimentation. Triangles are adaptively subdivided (Figure 2) in order to ensure, that no water particle can touch a triangle without influencing the erosion of at least one of its vertices. One of the obvious advantages of such a mesh is that any surface can be easily represented. There are however some important issues to deal with. If some vertices are moved further apart, then some others are moved closer together. This results in an oversampled mesh and while it doesn't pose any problems to the correctness of the simulation, it can significantly reduce the performance. Smooth merging of triangles without causing sudden changes in terrain geometry is yet to be implemented in this solution. The mesh can become degenerated when two surfaces get close enough to penetrate each other. This issue was solved in [Mul09], but can be difficult in case of an almost unconstrained mesh with arbitrary resolution used in the presented paper. A simple solution requires the previous problem to be addressed first. However, this degeneration seems to occur only in the last stages of a meander's evolution, so while it certainly limits the simulation, it is still possible to achieve decent results. Additional issues occur if the vertices defining consequent triangles on the river bed are at

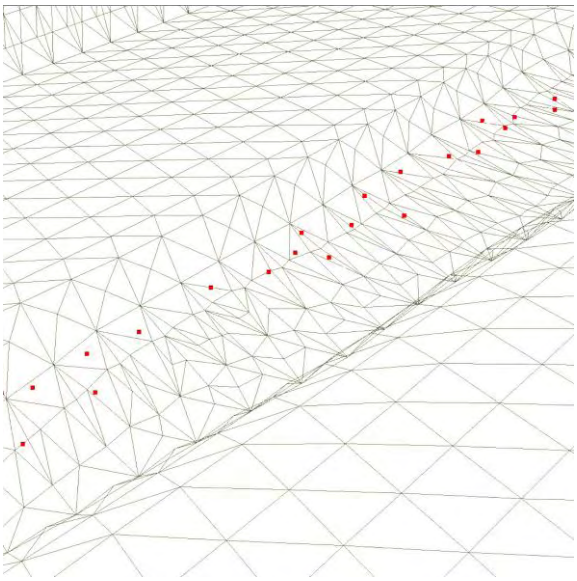


Figure 2. Terrain represented using a triangle mesh. The river channel is adaptively subdivided.

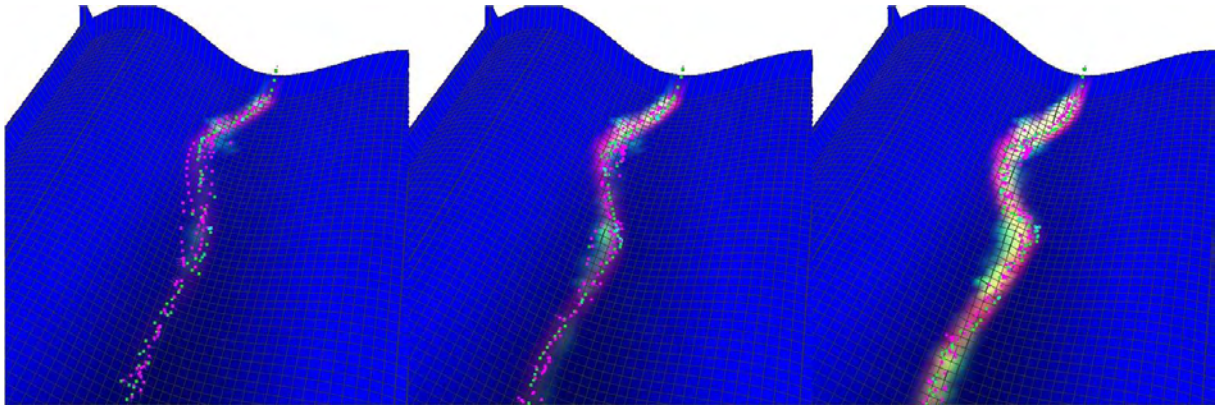


Figure 3. Evolution of a meander on a heightmap terrain (green particles – water; violet particles – sediment; green terrain – deposition; red terrain - erosion). The initial bends are due to the simulated Coriolis force and shape of the terrain. The positive feedback loop ensures the growth of the meander, which is unfortunately slower and slower, until it stops completely. It should also be noted, that the bend closest to the source is the most prominent, which is undesirable.

significantly different distances from the flow of water. This results in irregular erosion, which produces uneven bed surface which significantly slows the flow of water, creating unwanted small ponds. One solution is to introduce denser subdivision, which results in poorer performance. Other option which was also used was to perform smoothening of the eroded terrain after each 10 simulation steps.

The amount of eroded or deposited material is proportional to the magnitude of the displacement and the area supporting the displaced vertex. The deposition is always upwards and the erosion is assumed to be mainly downwards. Therefore the area is calculated as a sum of areas of all the triangles to which the given vertex belongs, projected on the horizontal plane.

Usually a hard bedrock or cohesive soil is harder to erode than fresh sediment. In order to take into account this feature, each vertex stores the thickness of a "soft layer". It indicates what part of the vertex displacement is due to the accumulation of material that was eroded elsewhere. If this value is larger than 0, we assume that we are eroding soft, fresh sediment. The introduced earlier material softness s of the base terrain is set to 1, while s of this layer is controlled by the user. While the thickness of the soft layer is increased by deposition, it is decreased both by erosion and time. The latter is introduced to simulate the hardening of the sediment into rock. The soft layer prevents sedimentation in places which are being constantly eroded and facilitates the transport of the material.

In order to facilitate the performance of adaptive subdivision, area calculation and collision detection, vertices are indexed in a 3D grid and contain information about all their neighbors and all the triangles they belong to.

4. SIMULATION SOFTWARE

The simulation software was written in C++ and uses FLTK library for user interface, OpenGL for visualization and OpenMP for parallel computations, so that it can take advantage of the modern multi-core processors. These libraries were chosen to enable the application to be compiled and run both on Windows and Linux systems. Water simulation, sediment transport and terrain representation were carefully separated in order to enable easy swapping of different terrain implementations. The code uses lambda functions introduced in the new C++11 standard. While they do not have any noticeable impact on performance, they greatly reduced the time needed to implement and test various approaches without compromising the introduced separation.

Water sources and sinks can be added, modified or removed before and during the simulation. SPH, erosion and deposition parameters can also be adjusted in run-time.

5. RESULTS

First experiments were conducted in an artificial valley with cross-section in the shape of a flattened sinusoid. Meandering in such a terrain occurs naturally if the stream of water is not perfectly aligned with the valley. However, many rivers meander on flatlands and so further experiments were conducted on a completely flat surface (4096 x 768 units in size) surrounded by barriers from 3 sides. In order to enforce the flow of water, the surface was tilted towards the side without any barrier. An artificial source producing one particle per second at a random position within a radius of 10 units was placed on the top. Water and sediment that fell from the surface after reaching its lower end was removed by an artificial sink. The created system was thus open. The CM factor regulating the force induced by Coriolis effect was set to 0.005. SPH radius was 25 units. Other user-controlled variables (over 20 in total, their complete description is beyond the scope of this article) were different in consequent experiments and sometimes tweaked in run-time.

The simulation contained usually around 300 SPH particles and a similar - sometimes slightly larger - amount of sediment particles. One step of the simulation took around 0.02s on Intel Core 2 Quad 2.66 GHz with 4GB of RAM. The approximate time spent in different subsystems is as follows: drawing ~ 1%, erosion ~ 25%, adaptive subdivision of terrain ~ 5%, SPH simulation ~ 30% (~25% is spent in calculating collisions with ground), sediment transport and deposition ~30%. Precise percentages depend strongly on the state of the simulation.

The artificial river bends, the outer banks are intensively eroded and the deposition occurs mainly on the inner ones. Meanders start to form and then grow (Figure 4). Dense subdivision is visible within the river channel, especially on the outer banks. The horizontal shape of the river looks convincing. However, the cross-section reveals one unwanted feature – the smoothing of the eroded terrain introduced to ensure the undisturbed flow of water causes the river channel to be too flat, lowering the bed near the inner banks (where most deposition occurs) and raising the bed near the outer ones (Figure 5).

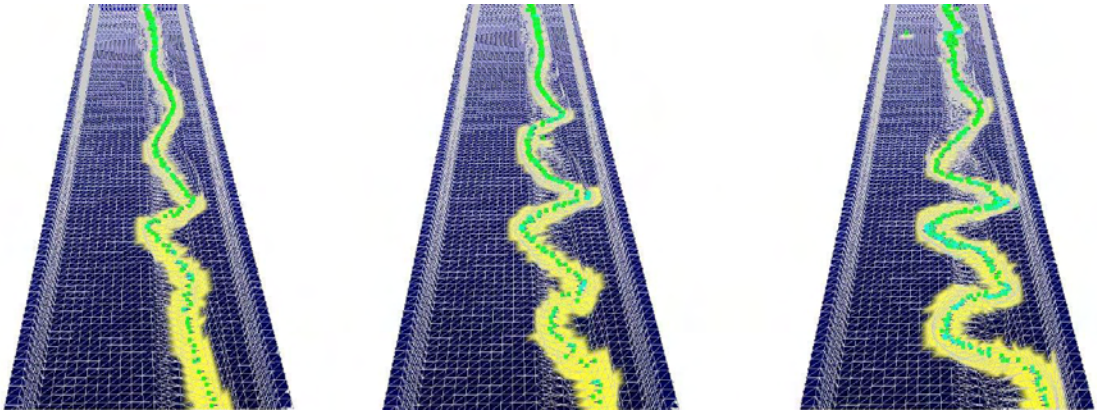


Figure 4. Meanders generated using the proposed method using a triangle mesh terrain representation. Green particles denote water and the yellow area corresponds to the river channel. It should be noted that the curves started to form due to the introduced Coriolis effect on an initially flat surface. The further downstream, the larger the meanders are, which gives them a plausible appearance.

6. CONCLUSIONS

The main goal of the experiments – to produce growing meanders – was achieved. Introduction of the simplified Coriolis force lead to satisfactory results. Triangle mesh seems to be the best choice for this type of simulation, but there are certain issues that require more work. In order to ensure the proper shape of the cross-section, the smoothing algorithm needs to be refined or completely replaced by a better solution with a possibly low negative effect on the performance. An effective solution to the mesh degeneration must be implemented in order to enable the full evolution of meanders up to the formation of billabongs. Additional tests on more complicated terrains should be conducted in order to find and resolve possible issues.

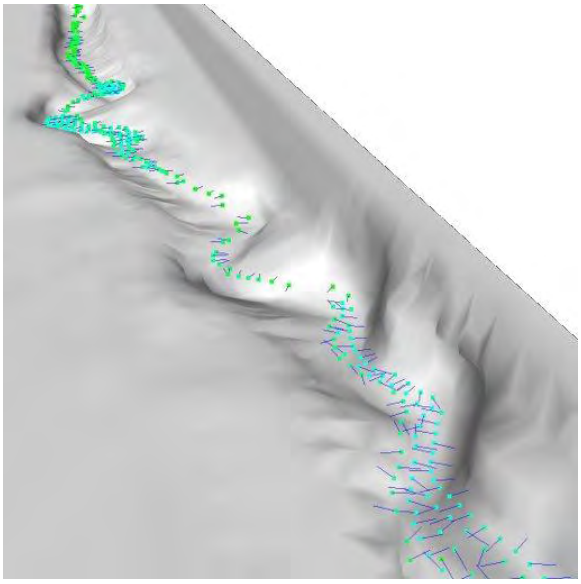


Figure 5. Close-up of the river channel (terrain surface is rendered using Gouraud shading with a single light source at the camera's position). The bed is too wide and too flat.

After refining the existing functionality, certain features will be added. At this moment the only sources of water are artificial and placed by the user. This results in the amount of water being approximately constant along the channel's length, while natural rivers tend to be small at the source and then grow larger due to additional supply from tributaries. Such tributaries should be automatically generated based on a rainfall simulation, probably similar to [Teo08]. The present method assumes the same material softness for the entire base terrain. It was shown in [Kur11b] and [Ben01] that introduction of several, possibly intersecting, layers of terrain with different parameters may significantly enhance the results of erosion. A similar feature should be implemented using the triangle mesh.

The SPH simulation was planned to be migrated to a GPGPU solution like OpenCL or CUDA. However, the solver uses just a few percent points of the processing time, so the expected benefits would be negligible. Possible performance related optimizations (and potential porting to GPGPU) should concentrate on the terrain representation and collision detection instead.

The proposed method has also an interesting feature compared to purely random algorithms – the emergence and growth of the meanders is a continuous process, which could be presented to the end-users as a feature. "From Dust" is a game in which the player assumes the role of a god and achieves the mission objectives by shaping the landscape using nature elements. One of these elements is water, which erodes the terrain. Adaptation of the proposed method for a direct use in a game environment would require a significant amount of work, but should be possible and could enhance the gameplay.

7. REFERENCES

[Bel05] Belhadj F. and Audibert P.. Modeling Landscapes with Ridges and Rivers. Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2005, 2005.

[Ben01] Benes B. and Forsbach R.. Layered Data Representation for Visual Simulation of Terrain Erosion. IEEE SCCG2001 Budmerice, Slovakia, 2001.

[Ben06a] Benes B.. Hydraulic Erosion: A Survey. Invited paper to SCCG 2006, ACM SIGGRAPH, 2006.

[Ben06b] Benes B., Tešinsky V., Hornys J. and Bhatia S.K.. Hydraulic Erosion. Computer Animation and Virtual Worlds 17(2), 2006.

[Ben09] Kristof P., Benes B., Krivanek J. and Stava O.. Hydraulic Erosion Using Smoothed Particle Hydrodynamics. Proceedings of Eurographics 2009 vol. 28 No.2, 2009.

[Bez10] Bezin R., Peyrat A., Crespin B., Terraz O., Skapin X. and Meseure P.. Interactive hydraulic erosion using CUDA. Proceedings of the 2010 international conference on Computer vision and graphics: Part I, 2010.

[Bro06] Brosz, J., Samavati, F. and Sousa, M.. Terrain synthesis by-example. Advances in Computer Graphics and Computer Vision International Conferences VISAPP and GRAPP 2006, 2006.

[Cen09] Puig-Centelles A., Varley P.A.C. and Ripolles O.. Automatic Terrain Generation with a Sketching Interface. Proceedings of the 17th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG '09), 2009.

[Chi05] Chiang, M.Y., Huang, J.Y., Tai, W.K., Liu, C.D. and Chiang, C.C.. Terrain synthesis: An interactive approach. Proceedings of the International Workshop on Advanced Image Technology, 2005.

[Ein26] Einstein, A.. The cause of the formation of meanders in the courses of rivers and of the so-called Baer's Law. Read before the Prussian Academy, January 7, 1926, published in Die Naturwissenschaften, Vol. 14 (English translation in "Ideas and Opinions," by Albert Einstein, Modern Library, 1994), 1926.

[Gos10] Goswami P., Schlegel P., Solenthaler B. and Pajarola R.. Interactive SPH Simulation and Rendering on the GPU. Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2010.

[Kel88] Kelley A.D., Malin M.C. and Nielson G.M.. Terrain simulation using a model of stream erosion. Proceedings of SIGGRAPH '88, 1988.

[Kur11a] Kurowski M.. Modelowanie terenu na bazie symulacji erozji z wykorzystaniem deformowalnych wokseli. Zeszyty Naukowe Wydzialu Elektroniki, Telekomunikacji i Informatyki Politechniki Gdanskiej, Proceedings of WGK 2011 vol. 1, 2011.

[Kur11b] Kurowski M.. Modelowanie terenu 3D z jaskiniami inspirowane erozją. Zeszyty Naukowe Wydzialu Elektroniki, Telekomunikacji i Informatyki Politechniki Gdanskiej, Proceedings of ICT Young 2011 vol. 1, 2011.

[Mul03] Müller M., Charypar D. and Gross M.. Particle-based fluid simulation for interactive applications. SCA '03 Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2003.

[Mul09] Müller M.. Fast and Robust Tracking of Fluid Surfaces. Proceedings of ACM SIGGRAPH / EUROGRAPHICS Symposium on Computer Animation (SCA), 2009.

[Mus02] Ebert D.S., Musgrave F.K., Peachey D., Perlin K. and Worley S.. Texturing and Modeling, Third Edition: A Procedural Approach. The Morgan Kaufmann Series in Computer Graphics, 2002.

[Nal97] Nalder G.. Aspects of Flow in Meandering Channels. Transactions of the Institution of Professional Engineers New Zealand: General Section Volume 24 Issue 1, 1997.

[Nei05] Neidhold B., Wacker M. and Deussen O.. Interactive physically based Fluid and Erosion Simulation. Proceedings of the Eurographics Workshop on Natural Phenomena, NPH 2005, 2005.

[Pru93] Prusinkiewicz P. and Hammel M..A Fractal Model of Mountains with Rivers. Proceedings of Graphics Interface'93, 1993.

[Ram99] Rameshwaran P., Spooner J., Shiono K., and Chandle, J.H.. Flow Mechanisms in two-stage meandering channel with mobile bed. Proceedings of IAHR Congress in Graz, Austria, 1999.

[Sme09] Smelik R.M., Kraker K.J., Groenewegen S.A., Tutenel T. and Bidarra R.. A survey of Procedural Methods for Terrain Modelling., CASA Workshop on 3AMIGAS, 2009.

[Sta08] Stava O., Benes B., Brisbinn M. and Krivanek J.. Interactive Terrain Modeling Using Hydraulic Erosion. Eurographics/SIGGRAPH Symposium on Computer Animation, 2008.

[Teo08] Teoh T.S.. River and Coastal Action in Automatic Terrain Generation. Proceedings of the International Conference on Computer Graphics and Virtual Reality'08, 2008.

[Teo09] Teoh T.S.. RiverLand: An Efficient Procedural Modeling System for Creating Realistic-Looking Terrains. ISVC '09 Proceedings of the 5th International Symposium on Advances in Visual Computing, 2009.

[Zho07] Zhou, H., Sun, J., Turk, G. and Rehg, J.. Terrain synthesis from digital elevation models. IEEE Transactions on Visualization and Computer Graphics 13,2007.

[Van11] Vanek J., Benes B., Herout A. and Stava O.. Large-Scale Physics-Based Terrain Editing Using Adaptive Tiles on the GPU. IEEE Computer Graphics and Applications November/December 2011, Vol 31, No 6, 2011.

Parallel Treecut-Manipulation for Interactive Level of Detail Selection

Daniel Schiffner
Goethe Universität
Robert-Mayer-Str. 10
Germany, 60054, Frankfurt (Main)
dschiffner@gdv.cs.uni-frankfurt.de

Detlef Krömker
Goethe Universität
Robert-Mayer-Str. 10
Germany, 60054, Frankfurt (Main)
kroemker@gdv.cs.uni-frankfurt.de

ABSTRACT

We present a dynamic system that allows to alter the Level of Detail (LOD) of a treecut-based object. The adaptation and selection is made in a parallel process which avoids stalling or locks because of expensive calculations and LOD changes. We present a method to control the exchange between the independent threads. Based on this separation, we present multiple strategies to perform the LOD-selection for point-based representations.

Keywords

Level Of Detail, Parallel LOD-selection, LOD-strategies, Thread Management.

1 INTRODUCTION

Level of Detail (LOD)-techniques are required in today's rendering environments to assure interactivity because of the ever growing number of primitives used [Hol11]. The selection of a LOD-representation, for example, can be based on the current view or object-related properties. However, these selections may require expensive computations, and thus, discrete LODs are preferred over continuous methods. We address this issue and present a system to allow parallel LOD-selection.

This LOD-selection can be derived using different strategies. These range from a simple recursive algorithm up to a priority-based selection. Using a perceptual metric in combination with a prioritization, the visual quality of an existing representation is preserved with respect to the human visual system. As the necessary calculations made by a perceptual metric can be expensive, the LOD-selection is performed in a parallel thread. So, stalling of the rendering is reduced to a minimum.

In this work, we describe our point-based rendering system and show how to manage the individual threads. Furthermore, we present multiple LOD-strategies that evolve an existing representation using only local operations.

Following to this introduction, we will give an insight into related work and then present our framework. This includes the synchronization of the evaluation as well as the LOD-strategies. We present performance and visual results that were achieved with the proposed framework and conclude with an outlook regarding future work.

2 RELATED WORK

Several methods to create a LOD-hierarchy exist. Typically, these levels are created by the hand of a designer who may be supported by a reduction algorithm. These pregenerated LODs are then used during rendering and are exchanged by some kind of metric.

It is possible to avoid the generation of a hierarchy or pregenerated LODs to reduce the amount of stored information. Especially interesting in this context are progressive representations. These produce intermediate solutions and are not restricted to fixed, i.e. discrete, levels. For this reason, these methods are referred to as continuous LOD. Hoppe [Hop96] presented a mesh-based reduction method, which was extended to point-based representations by [Wu05].

To create the individual levels, the reduction methods can exploit geometric information to increase details. [Gar97], for example, apply an error metric to increase the quality of the reduced versions. If such an algorithm is applied sequentially and the results are stored, a hierarchy is created. The current representation is defined by a vertex front or a cut / treecut.

Instead of generating details, the vertex front can be altered to select the representation based on the current hierarchy. Schiffner and Krömker [Sch10] use a treecut to adapt the representation by prioritizing nodes with high curvature. A similar approach is presented

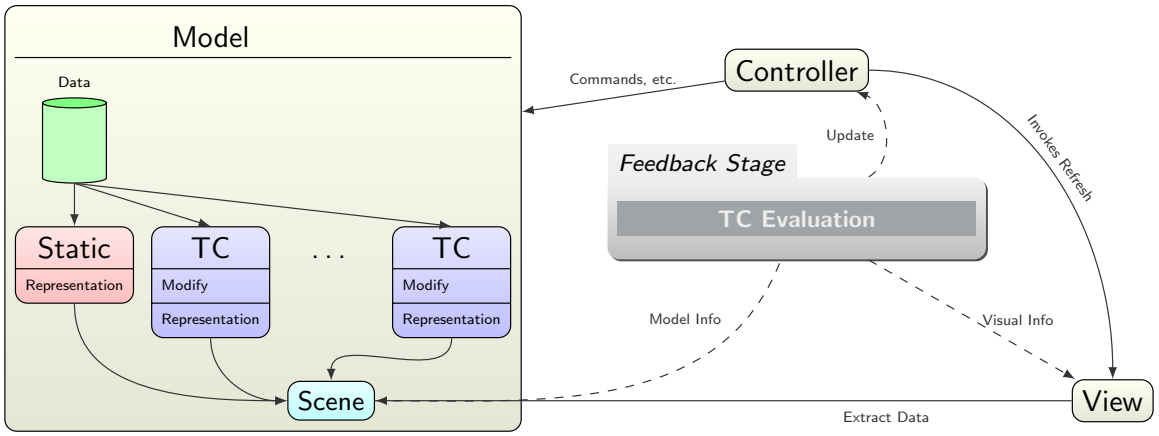


Figure 1: The design of our framework is based upon a Model-View-Controller (MVC) pattern. We introduce a Feedback Stage, which consists of a LOD-strategies and generates the new representation of the *TreeCut*. This component extracts information from both the Model and the View. Changes are relayed via the Controller.

by [Car11], where an octree cut is used to select data for visualization of large data sets. Both methods avoid a retraversal of the hierarchy and preserve the current vertex front, similar to Progressive Meshes [Hop96] or Progressive Splats [Wu05]. Both cuts includes a method to alter the current distribution. This idea will be used as a so-called LOD-strategy within this work.

Multi-threaded or parallel applications for rendering often focus on splitting the current representation into multiple viewports. Applications here range from Global Illumination [Hol11] to efficient large data set visualization [Gos12]. Recently, Peng et al. [Pen11] presented an approach to generate large crowds by parallel processing the individual models. Similar to our work, a scene is optimized for interactive renderings. They, however, use a fixed evaluation method.

3 SYSTEM DESIGN

As common in graphics applications [Shi10], our framework is based upon a Model-View-Controller (MVC) pattern. The Controller invokes the changes of the Model, i.e. the *TreeCut* [Sch10], which holds the current LOD representation. The View uses this current representation to derive a visual output using a graphics API.

We extend this default pattern by introducing a Feedback Stage (see figure 1). It is similar to an observer component, but with the ability to influence the Controller. Thus, the Feedback Stage can alter the Model which results in a different View. The Feedback Stage utilizes a strategy to apply the necessary LOD-operations. Therefore, the strategy may need to extract information from the rendered scene. This is visualized by the connection between the Feedback Stage and the View in figure 1. In the following, we will refer to the extraction of information and application of the strategy as an iteration.

Thread Management

We separate the Feedback Stage from the Controller, as it is an independent component. As it has access to all information required, it will be executed in parallel to the default rendering. During evaluation, the rendering of the old representation is continued. Due to the parallelization of both processes, some kind of synchronization needs to be included to avoid deadlocks or race-conditions.

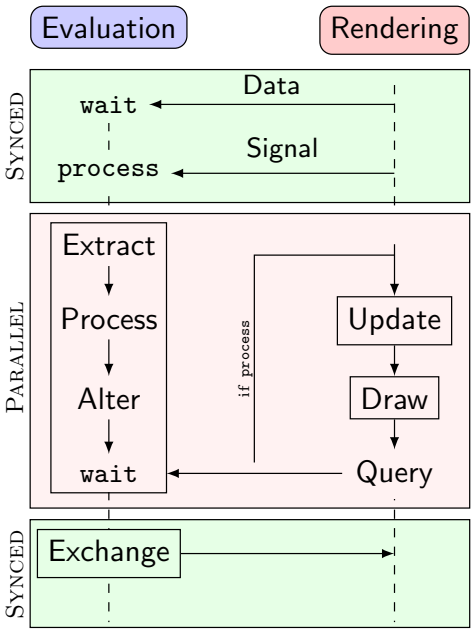


Figure 2: The processing sequence used to control the evaluation and rendering threads. While the evaluation is processing, the rendering thread displays the current LOD-version. Once the evaluation has completed, the data is exchanged and the LOD is updated gracefully. The query from the rendering thread is made without lock. This conditionally triggers the exchange of a new representation, which must be performed synchronized.

We propose a synchronization-strategy based on two states that are queried by the rendering thread: `wait` and `process`. This allows to add the evaluation with only small changes to the rendering code. The rendering thread only has to query for a new representation, while the Feedback Stage will handle the complete strategy evaluation and LOD-selection. The performed steps are visualized in figure 2.

We leverage the fact that the *TreeCut* is only represented with an index-list. The derivation of a new LOD thus only requires to generate a new index-list, which will be swapped or blended with the current one. This minimizes stall and flicker once a new representation is available. Only a pointer, or the VBO id, needs to be replaced. No copying of this data is performed during synchronization.

On initialization of the evaluation thread, the state is set to `wait`. In this state, all data can be accessed safely from the rendering thread. Here, no locking the data is required as no processing is performed. Only in this state the data will be exchanged. As stated before, the evaluation will generate an index-list, which can be swapped with the current one used for display.

If a new representation is requested, because the scene has changed or is considered invalid, the rendering process issues an update request to the evaluation via a signal. The evaluation thread is then set to the `process`-state. The rendering continues displaying the old representation as long as the evaluation is generating an updated version. After rendering a single frame, the evaluation is queried.

When starting to generate a new representation, the evaluation extracts the required data. This includes to copy the current index-list used by the rendering thread. As this is an read-only operation, no lock is required. LOD-strategies may need to aquire additional data from the View or the Model which can also be copied without a lock.

After completion of a single iteration, the evaluation thread will change its state to `wait`. As it is possible that the current representation is optimal for the applied strategy, a flag is used to indicate this case. This also allows to accelerate the LOD-strategies as they may terminate prematurely.

When the evaluation is in `wait`-stage again, the Exchange is executed. The Exchange does not cause a race-condition in both threads, because neither the rendering nor the evaluation requires access to the crucial data at this time. Additionally, we only require to exchange, i.e. swap, the used index-list if a new representation has been generated.

The Feedback Stage can be invoked again directly after the completion of the iteration. No additional updates to the Feedback Stage are required, as it extracts the current information in parallel.

4 LOD-STRATEGIES

Once a *TreeCut* has been established, only two core operations are applied (`refine`, `coarse`), which represent the changes in the detail. To alter the representation in a global manner, we apply LOD-strategies that are based solely on the current cut.

We include a threshold value to control the application of the individual operations. This counteracts repetitive `refine`s or `coarse`s of nodes. We, hereby, mean that a node is `refine`d in an iteration while it is `coarse`ned in the next.

In the following, we will present three different types of strategies: An optimization, a bucket-based approach and a recursive traversal of the hierarchy. The latter differs from the first ones because it operates on the complete LOD-hierarchy instead. It is included to show the universal applicability of the proposed thread management.

Optimization Strategy

The optimization LOD-strategy evaluates the current cut and applies a partial sorting based on a priority value, similar to [Car11]. This strategy requires some kind of limitation regarding the cut-size, e.g. a maximal node count. The priorities of the parent nodes should to be larger than their children to avoid artifacts. Otherwise, a parent node, i.e. a coarser representation, is favoured over a more detailed one.

During the Extract in the evaluation thread (refer to figure 2), the priorities are aquired. In our implementation, we use the curvature from the cut-nodes as priorities. The partial sorting is applied by iterating the complete cut and storing only the nodes with highest and lowest priority.

The algorithm selects nodes with highest priority for `refine`ment, while nodes with lowest priority are

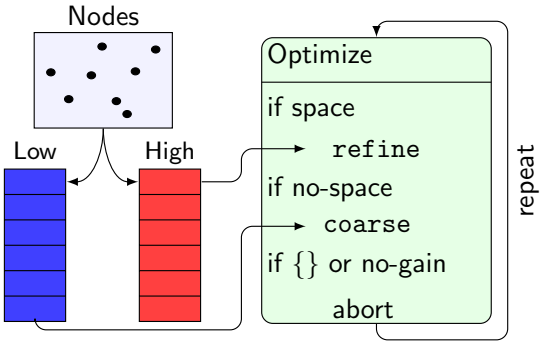


Figure 3: The optimization LOD-strategy for *TreeCut*-evaluation. Only the nodes with highest and lowest priority are processed. This accelerates the evaluation as it reduces the theoretical time complexity [Car11]. The most important ones are `refine`d, while the least important ones are `coarse`ned. This strategy requires a maximal node count to be applied

coarsened in the representation. A coarse frees space for further refinement with more important nodes. The operations performed by this strategy are visualized in figure 3.

As the partial sorting size can be considered constant during run-time, e.g. the size is not changed during an iteration, a linear time-complexity is given: $O(n \log k)$ with k being the constant partial sorting size and n the size of the current cut.

The threshold is defined as the minimal gain in priority required when altering the *TreeCut*. Therefore, the primitives in the sorted sequences (low and high) are compared before a coarse is applied. A refine is always executed as long as space is available.

Bucket-based Strategy

The second strategy assigns a target bucket to each node within the cut. The strategy alters the *TreeCut* to match a certain distribution as closely as possible. This strategy requires a method that determines the target bucket for a node.

An example for application is the generation of a stippling-like appearance of an object. The target bucket for each node is derived by using the illumination at the current node's location. The darker the current location, the more nodes are used within this region, i.e. the hierarchy is refined. The node is coarsened if a lighter representation is required.

In figure 4, an exemplary application of the LOD-strategy is shown. For each node, a target bucket is calculated. If this bucket differs from the currently assigned bucket, the delta is used to determine the according cut-operation. In the figure, a + denotes a positive delta and a refine needs to be applied, a - is a coarse. The 0 is the special case, that the node already has the correct bucket and no operation is necessary.

After the buckets have been calculated for all siblings, the operations are validated. As in the optimization LOD-strategy, a refine has higher precendence than a coarse. For this reason, the left branch is expanded in figure 4.

Special care has to be taken, if a coarse-operation needs to be applied. The parent node needs be inspected as well. The operation is only executed, if the bucket of the parent does not invalidate it. A small example will illustrate this scenario.

In the right branch of the tree in figure 4, a coarse needs to be applied. Therefore, the parent node is inspected (visualized by the question mark). In this case, the target bucket for the parent does not have a different delta (it is 0), i.e. it does not invalidate the operation. Thus, the coarse can be applied safely. The same applies, if the delta would be negative. If the parent would have a positive value, the node would be expanded in the next iteration. This would invalidate the operation and introduce a flicker into the represenation and the coarse is not executed.

As each node within the cut is evaluated, a linear time complexity is given: $O(n)$ with n being the cut-size.

For this LOD-strategy, the threshold is defined as the minimal delta that is required to force a coarse. We have achieved good results by a threshold of 0.

Recursive Strategy

The last strategy evaluates the complete LOD-hierarchy instead of the current cut. This method is inspired by the QSplat rendering system [Rus00]. Starting at the root node, the new cut is defined by the individual nodes when aborting the recursion. This abort is either due to culling, small splat area, or when no further refinement is possible, i.e. a leaf node is reached.

For this method it is required to additionally store the complete hierarchy, which is not the case for the other two LOD-strategies. During the Extract-step, this hierarchy is mapped to be accessible. The evaluation then starts the recursive traversal on a plain index-list.

The worst time complexity of this algorithm is $O(N)$ where N is the number of nodes within the tree. As the abort criterion includes culling, an acceleration is achieved, which results in an average logarithmic time complexity for large objects.

As opposed to the other two methods, the recursive strategy generates a new cut instead of manipulating an existing one. Thus, the definition of the threshold is not applicable to this strategy.

5 RESULTS

We tested the different evaluation strategies with our rendering system. We measured the rendering times and the overhead introduced by the usage of our system. The proposed LOD-strategies are compared to each other and the evaluation times in dependency of the original primitive count and the current count will be given as well.

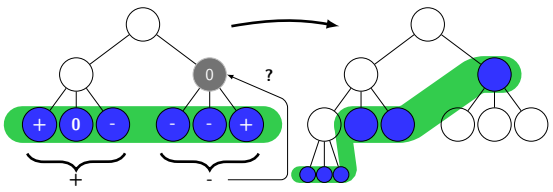


Figure 4: The bucket-based strategy for *TreeCut*-evaluation. Each node is assigned a target node. All siblings and the parent define the operation to be applied. If a coarse-operation is requested, the parent node is inspected (indicated by the question mark). Only if the operation is considered save, it is executed.

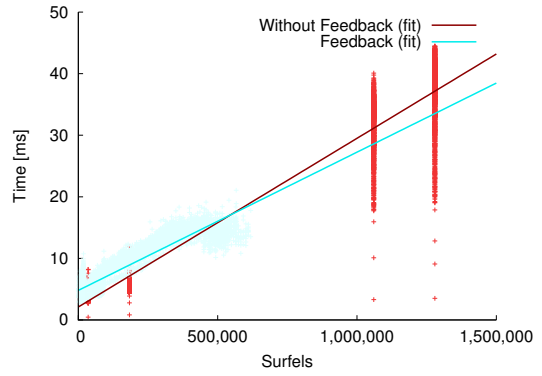


Figure 5: The performance impact when using the proposed Feedback System. The new representation generated is swapped from the evaluation thread to the rendering. Note that the increase is not required every frame, but only when an iteration has been completed. In our prototype, no changes are made to the representation during rendering. The average overhead is the difference between the fitted lines.

A sequential comparison is not included using the proposed strategies, but can be derived easily by summing up the rendering and evaluation times.

Additionally, we present some visual outputs generated by our renderer. For all renderings, we use a point-based rendering method that utilizes the Phong Splatting technique presented by [Bot05].

As noted before, we use the curvature as the priority value in case of the optimization LOD-strategy. The curvature identifies important regions on the surface of the object, and detail is preserved in regions where the surface changes. This was also presented in [Sch10] and [Lee05]. We preprocessed the curvature and included it within the LOD-hierarchy. Additionally, we assure that parent nodes have a higher curvature value than their children. During rendering of the scene, the maximal node count is set to be idendtical to the count given by the recursive LOD-strategy.

The bucket-based strategy uses the illumination information to derive a target bucket for each node. We use a fixed splat size for each node, and so create a stippling-like appearance of an object.

Finally, the recursive method implements the QSplat hierarchy and enables the basic QSplat method to be rendered efficiently using the Phong Splatting technique [Bot05] without further adaptation.

Time Measurements

Our prototype is written in `C++` and openGL. The tests were made with a Intel i5 with 3.47 GHz, 8.0 GB RAM and a nVidia GeForce 260 GTX with 896MB RAM. The graphics in figure 5 show the overhead introduced due to the exchange of the newly generated LOD-version after an iteration has been completed. Note that this increase is only generated if the evaluation is in `wait`-stage and a new LOD-version was created. The overall time falls below the version without the Feedback Stage (labeled Without Feedback) because rendering is accelerated and less primitives are required.

The graphs in figure 6 show the performance of the proposed strategies. We tested each strategy with multiple objects that are drawn in a predefined scene. During rendering, only one object and one light source is used. Both are rotated and moved to assure a large number of update request for the LOD-strategies. The objects are taken from the Standford 3d repository [3DScan]. As the same scene is used for all objects and LOD-strategies, the aquired evaluation times are comparable.

We omit information of the transfer of the data from the evaluation thread to the rendering thread as the generated data is only swapped.

As expected, both *TreeCut*-based strategies perform with linear time complexity. The bucket-based (refer to figure 6a) LOD-strategy does not change the size of the *TreeCut* as much as in the optimization LOD-strategy (refer to figure 6b). This is due to the fact that the bucket-based strategy is not limited by a maximal node count.

The optimization LOD-strategy allows to include any priority into an existing hierarchy. The evaluation remains linear despite the performed sorting. In the shown case, 8k elements are sorted.

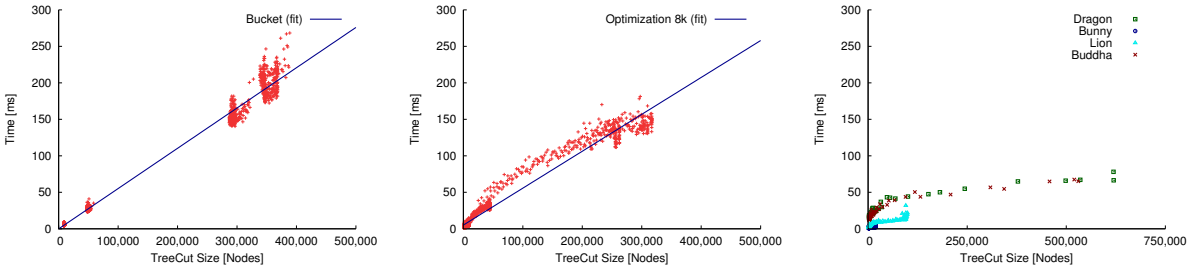
The recursive strategy applies the QSplat traversal presented by [Rus00]. The abort criterion includes backface culling, and for this reason, multiple nodes or surfels can be rejected early. This results in an overall acceleration of the traversal. For simple objects, the gain is not as large as with objects with higher geometric complexity. However, the strategy does not allow fine-tuning of a single representation. Only the recursive algorithm can be altered.

Visual Results

Some results achieved with the proposed LOD-strategies are shown in figure 7. A directional light source is used for illumination.

In figure 7a, we applied the bucket-based strategy along with an illumination-based target bucket function. We determine the target bucket by weighting the current illumination with respect to the depth of the node and maximal depth of the tree. This creates a stippling-like appearance of the drawn object. We enhanced Phong Splatting technique for the bucket-based approach to generate both a closed surface and equal-sized surfels.

The visual quality of the rendered version depends mainly on the used hierarchy. We enhanced the generation by using a node as parent instead of the average



(a) Bucket-based LOD-strategy performance graph.

(b) Optimization LOD-strategy performance graph.

(c) Recursive LOD-strategy performance graph.

Figure 6: The performance graphs of the proposed LOD-strategies. The values are given for an average single iteration. The LOD-strategies have been applied to mulitple objects with varying sizes. The first two offer linear time complexity, but the priority strategy has a larger overhead due to the required partial sorting. The recursive strategy is able to fast reject large portions of the hierarchy, which results in logarithmic time.

as proposed by [Rus00]. This suppresses motion of surfels when changing the LOD. Also, the more leaf nodes are available, the better the dark regions can be displayed.

A result generated with the optimization LOD-strategy is shown in figure 7b. The surfels are prioritized by the local curvature and surfel-size. This preserves details at regions where the object's surface is changing. Larger surfels are used in flat regions resulting in a reduction of the number of used surfels. In the shown image, only 45k surfels (original 183k) are used. The surfel-size has been added to avoid generation of too large splats that could mask detailed areas. This additional information is solely required for point-based representations.

Figure 7c shows the result created with the recursive LOD-strategy. Obviously, there is no difference in the visual quality compared to the original QSplat algorithm if plain splatting is used. However, the parallelization increases performance of the rendering. This is because the rendering can leverage VBOs and so avoids repetitive transfer of rendering data.

A higher visual quality is achieved by using the Phong Splatting technique. This can be used without any adaptation as the LOD-strategy is independent of the rendering. With the original QSplat, the multiple renderpasses of the Phong Splatting would require to traverse the hierarchy more than once, which would massively penalize the performance.

A rendering with the maximal available detail of a sample object (the Stanford lion) is depicted in figure 7d. It uses all 183408 leaf nodes and does not offer more details than the reduced versions (shown in figures 7b (45K) and 7c (87K)).

6 CONCLUSION AND OUTLOOK

Our approach increases the range of parallel processing existing LOD-hierarchies. The different LOD-strategies account for many scenarios, ranging from budget-based restrictions with perceptual optimization up to a bucket-based selection where nodes are assigned a specific level. Also, non-cut-based methods can benefit from the proposed system, which has been shown as well by including the QSplat algorithm.

As the object and selection is made in parallel, no stalling of the actual rendering occurs. In addition, the exchange between old and new representation can be made with blending to avoid flicker artifacts. The newly deduced LOD is not generated in advance, but created using information from the current representation. This allows a finer grained adaptation to a given scenario. Due to the design, the system can be included into existing LOD-management systems as a data provider for new LOD-versions.

All presented strategies have a low theoretical time complexity and show good performance in our prototype. The parallel processing of the data preserves interactivity of the rendering without being restricted to a fixed LOD-set. The synchronization is achieved by a simple query and thread-safe exchange is assured by design.

The LOD-strategies can be extended to account for information that is present in the scene. For example, the optimization strategy can include perceptual information acquired from the current scene. This increases the quality of the representation, while no new data needs to be generated. Especially interesting is the application of the *TreeCut* methods within the GPU to completely avoid transfer of data between CPU and GPU.

Yet, the system and the strategies are not optimal and need to be refined. Similar to other approaches, we plan to evaluate our system using many objects. The question arises, whether a centralized thread or a agent-based approach provides better results. Developers should be supported to decide which is the best for their scenario.

(a) Bucket-based strategy. The Stanford dragon is altered in dependency of the illumination.

(b) Optimization strategy. The surfels for rendering are prioritized by their curvature and surfel-size. This version uses 45k surfels.

(c) Recursive strategy. The surfels are generated by the QSplat algorithm (87k surfels).

(d) High quality rendering without the Feedback Stage (183k surfels).

Figure 7: Results generated with proposed LOD-strategies. A stippling like appearance can be created by determining the target bucket based on the current illumination. The second figure has been generated with the curvature and surfel-size as priority. The recursive splatting method presented by QSplat can be accelerated by employing a recursive evaluation strategy. The figure on the right shows the high detailed version of the Stanford lion.

We plan to include a complete perception model in the optimization LOD-strategy. In this case, a full 3d model, like [Sch11] or [Lee05], seems most suitable, as the perception information is extracted directly from the 3d data.

The bucked-based method does currently not include important properties like blue-noise [Hil01]. This information needs to be encoded within the hierarchy during generation of the LOD and has to be ensured during selection of the individual nodes as well. Also, the target bucket function needs to carefully select nodes for replacement.

Finally, the system itself needs be enhanced, so that the performance and the selection quality increases. We plan to extend it with environmental information, e.g. processing power or battery. This allows to selectively apply the LOD-selection on different hardware platforms, while being restricted to a universal, system-independent criterion.

7 REFERENCES

[3DScan] `http://graphics.stanford.edu/data/3Dscanrep/`.

[Bot05] Botsch, M., Hornung, A., Zwicker, M., Kobbelt, L. P. High-Quality Surface Splatting on Today's GPUs. Eurographics Symposium on Point-Based Graphics. pp.17-24. 2005.

[Car11] Carmona, R., Froehlich, B. Error-controlled real-time cut updates for multi-resolution volume rendering. Computers & Graphics. pp.934–944. 2011.

[Gar97] Garland, M., and Heckbert, P. S. Surface Simplification using quadric errormetrics. The art and interdisciplinary programs of SIGGRAPH 97. pp.209–216. 1997.

[Gos12] Goswami, P., Erol, F., Mukhi, R., Pajarola, R., Gobbetti, E. An Efficient Multiresolution Framework for High Quality Interactive Rendering of Massive Point Clouds using Multi-way kd-Trees. The Visual Computer 28. 2012.

[Hil01] Hiller, S., Deussen, O., Keller, A. Tiled Blue Noise Samples. Vision, modeling and visualization. pp.265–272. 2001.

[Hol11] Hollander, M., Ritschel, T., Eisemann, E., and Boubekeur, T. ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination. Computer Graphics Forum 30. pp.1233–1240. 2011.

[Hop96] Hoppe, H. Progressive Meshes. SIGGRAPH 96 conference proceedings. pp.99–108. 1996.

[Lee05] Lee, C. H., Varshney, A., Jacobs, D. W. Mesh saliency, Proceedings of ACM SIGGRAPH 2005. pp.659–666. 2005.

[Pen11] Peng, C., Park, S., Cao, Y., and Tian, J. A Real-Time System for Crowd Rendering: Parallel LOD and Texture-Preserving Approach on GPU. Lecture notes in computer science vol. 7060. pp.27–38. 2011.

[Rus00] Rusinkiewicz, S. and Levoy, M. QSplat: a multiresolution point rendering system for large meshes. SIGGRAPH 2000 conference proceedings. pp.343–352. 2000.

[Sch10] Schiffner, D., Krömker, D. Tree-Cut: Dynamic Saliency Based Level of Detail for Point Based Rendering. Sensyble 2010. pp.37–43. 2010.

[Sch11] Schiffner, D., Krömker, D. Three Dimensional Saliency Calculation Using Splatting. Sixth International Conference on Image and Graphics (ICIG). pp.835–840. 2011.

[Shi10] Shirley, P., Marschner, S. R., and Ashikhmin, M. Fundamentals of computer graphics 3rd edition. 2010.

[Wu05] Wu, J., Zhang, Z., and Kobbelt, L. P. Progressive Splatting. Eurographics Symposium on Point-Based Graphics. pp.25–32. 2005.

Animation of Water Droplets on a Hydrophobic Windshield

Nobuyuki Nakata
The University of Tokyo
5-1-5 Kashiwa-no-Ha
Kashiwa, Chiba
277-8561 Japan
nobnak@nis-lab.is.s.u-tokyo.ac.jp

Masanori Kakimoto
Tokyo University of Technology
1404-1 Katakura-machi
Hachioji, Tokyo
192-0982 Japan
kakimotoms@stf.teu.ac.jp

Tomoyuki Nishita
The University of Tokyo
5-1-5 Kashiwa-no-Ha
Kashiwa, Chiba
277-8561 Japan
nis@is.s.u-tokyo.ac.jp

ABSTRACT

Animation of water drops on a windshield is used as a special effect in advanced driving games and simulators. Existing water droplet animation methods trace the trajectories of the droplets on the glass taking into account the hydrophilic or water-attracting nature of the glass material. Meanwhile, in the automobile industry, usage of hydrophobic glass windshields has recently been a common solution for the drivers' clear vision in addition to cleaning the water with wipers. Water drops on a hydrophobic windshield behave differently from those on a hydrophilic one. This paper proposes a real-time animation method for water droplets on a windshield taking account of hydrophobicity. Our method assumes each relatively large droplet as a mass point and simulates its movement using contact angle hysteresis accounting for dynamic hydrophobicity as well as other external forces such as gravity and air resistance. All of a huge number of still, tiny droplets are treated together in a normal map applied to the windshield. We also visualize the Lotus effect, a cleaning action by the moving droplets. Based on the proposed simulation scheme, this paper demonstrates the motion of the virtual water droplets on the windshield of a running vehicle model.

Keywords

Water droplets, hydrophobicity, windshield, driving simulator, contact angle hysteresis

1. INTRODUCTION

Water flow on the window or windshield surfaces are commonly used as a rainy scene description in film works and other types of motion pictures. More recently, computer generated animations of water flow on the windshields are realized for advanced video games and driving simulators. Since the glass material has hydrophilic or water-attracting nature, water droplets move along irregular trajectories seeking for water-attracting places of the surface, as we often find on the windows in a rainy day. Most of the existing water droplet animation methods simulated these winding trajectories of the droplets.

In real driving situations, those water trajectories or water-film on the windshields due to the hydrophilicity seriously affect the visibility through the glass. To clear the water, mechanical wipers have been used since the beginning of the automobile history. In addition, as auxiliary measures, coating the windshield with water repellent material became a solution a few decades ago. In the year 2000, the first water-repellent finished windshield became commercially available. Nowadays such hydrophobic windshield products are widely used in the automobile market.

A large amount of research literature on the behaviour of water on hydrophobic surfaces is published in chemical and mechanical engineering fields. To the authors' knowledge, however, little work has been done on real-time simulation of water droplets sliding across hydrophobic windshields. In this paper, we address this problem and propose a solution consisting of several practical simulation models for use in games and driving simulators.

Water attracting or repelling feature of surface material should be quantified differently in two situations, static and dynamic. The static repellency has been investigated for a long time and the fundamentals have been established. For water droplet animation, knowledge on the dynamic repellency is more important, which is true in

engineering analysis of water-shedding phenomena on the windshield. While the dynamic water repellency includes a number of unexplainable phenomena, there are a couple of major factors and indicators characterizing the dynamic repellency. Those include contact angle hysteresis, falling angle, falling velocity, and falling acceleration.

The relationship between the contact angle hysteresis and the slope angle has long been investigated. In case of an ideal water droplet shape, the contact angle hysteresis is known to be in proportion to the falling angle.

The falling velocity and acceleration vary by the surface material even when the slope angle remains constant. Although the standard methods for evaluating and measuring the falling velocity/acceleration were not established until recently, it is known that the behaviour of a falling water droplet on the hydrophobic surface is explainable in terms of rolling and sliding.

In this paper, we take the knowledge on the dynamic repellency into account and propose a real-time animation method for water droplets on the hydrophobic windshield. As the water-repellent coated windshields become standard in the automobile market, our contribution is to provide video game and simulator developers with a means of reproducing realistic and harmonious motions of the water droplet cluster traveling across the hydrophobic windshield.

This paper is organized as follows. In the next section we introduce related work on both engineering analyses and animation techniques for water droplets. Then our proposed method is explained in a theoretical point of view in Section 3, followed by more detailed descriptions on the implementation and results in Section 4. Finally we give conclusions and future work in Section 5.

2. RELATED WORK

In the computer graphics field, several methods have been introduced for animating water droplets. Kaneda et al. [Kan93a] [Kan96a] proposed methods to describe the movement of the droplets by defining each droplet as a particle and move it with particle dynamics. Since the droplets travel seeking for water-attracting places, their trajectories on the glass surface form complex shapes. They also simulated these motions by a random walk method using random numbers [Kan99a]. Recently their method was implemented as a real-time simulator with a GPU computing technique [Tat06a]. Fournier et al. [Fou98a] depicted the trajectories of droplets using the mass spring model. None of the above methods took into account the hydrophobicity of the inclined surface since they assume hydrophilicity. Also, they do not incorporate air resistance against the water drops or rolling resistance of the drops.

Several researchers have developed fluid dynamics based methods for the water droplet simulation. Wang et al. [Wan05a] took into account surface tension, contact angle, and contact angle hysteresis. The surface tension is more dominant in a water droplet than in regular large-scale fluid forms. Thürey et al. [Thu10a] introduced the mean curvature flow, which is known as a motion equation for surface boundaries, and evaluated the phenomena caused by the surface tension more appropriately than Wang et al.

Zhang et al. [Zha11a] developed a faster computation method for droplets using the mean curvature flow without other fluid simulations. They ignored the internal fluid flow of the droplets but used the surface tension and other external forces to give deformation, collision and division to each droplet represented as a polygon mesh. They achieved 10-50 fps in the experiment with 10K-50K polygon mesh. However, due to the implicit method for the mean curvature flow computation, the stability of their solution depends highly on the mesh quality and the time step, and the performance optimization is limited.

In order to tackle the problem of the droplet motion on the hydrophobic surfaces, we need to understand dynamic repellency. The structure or the behaviour of the surface molecules are considered to be a source of the dynamic repellency. To figure out the behaviour, Hirvi et al. [Hir08a] simulated a droplet consisting of thousands of water molecules using a molecular dynamics calculation technique. Korlie [Kor93a] proposed a cluster model of quasi-molecular particles on a horizontal plane and introduced its dynamical equations which lead to the value of the contact angle of the cluster.

Analyses of real water droplets have been done by several research groups. For example, Sakai et al. [Sak06a] measured the velocity and the acceleration of a droplet sliding across water-repellent surfaces. Droplets are known to run down either rolling or slipping on the incline depending on the degree of hydrophobicity [Ric99a] [Suz09a]. Hashimoto et al. [Has08a] measured the relationship between the volume and the velocity of a windswept droplet.

We address the problem of dynamic water-repellency taking the contact angle hysteresis into account. In addition, we use the knowledge of the real water drop analyses to verify and compensate our results. We avoided using the fluid dynamics simulation, the mean curvature flow, or any type of molecular forces since they are not suitable for real-time visualization. Due to the computing load and

the time step limitations, those methods cannot handle sufficient number of droplets on a car windshield.

In our method, each droplet is represented as a mass point or a particle. Thus, we are able to incorporate additional forces into the real-time simulation loop; air resistance against the water droplets and viscous dissipation which acts as a rolling resistance of each drop. Although these forces are crucial factors for the fast movement of water drops, they have not been fulfilled in the previous methods [Wan05a] [Thu10a] [Zha11a].

Particle dynamics are common in the real-time simulation field. They are widely adopted in games and interactive applications. Real-time physics engines in the market are equipped with features of particle dynamics and rigid body dynamics including collision detections as fundamental functions. We implemented our method on top of a game engine 'Unity' and added unique behaviours of water droplets running slowly or quickly, or staying on the hydrophobic surfaces.

3. A PRACTICAL MODEL FOR WATER DROPLETS ON HYDROPHOBIC WINDSHIELDS

3.1 Water Droplet Geometry

When a droplet is on a solid surface, the contact angle is defined as the angle between the solid surface and the droplet surface. The contact angle is determined by the Young equation, which describes the balance of three surface tensions, as shown in Equation (1).

$$\gamma_L \cos\theta = \gamma_S - \gamma_{SL}, \quad (1)$$

where, θ is the contact angle, γ_L is the surface tension of the water droplet, γ_S is the surface tension of the solid, γ_{SL} is the boundary tension between the water and the solid (Figure 1).

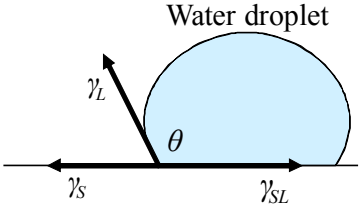


Figure 1. Contact angle and tensions of a water droplet.

When the radius of the droplet on hydrophobic surfaces is less than the radius of capillary (2.8mm), the surface tensions are the dominant factors of the water drop shape. Thus the droplet forms a near spherical geometry. Meanwhile, the contact angle of the glass becomes 90°-100° when it is coated with commercially available repellent material.

Based on the above two observations, we assume that each rain droplet is rendered as a hemisphere. In practice, the geometric shape is basically a disc-like plane and the normal vectors for refraction are controlled to make it look hemisphere. Details are described in Section 4.3.

3.2 Contact Angle Hysteresis

When a thin pipe is inserted into water, the water level in the pipe is raised by the capillary action. This is caused by a force called the capillary force which operates along the triple boundary line among the water, the solid and the air. The capillary force is determined by the Young-Laplace equation.

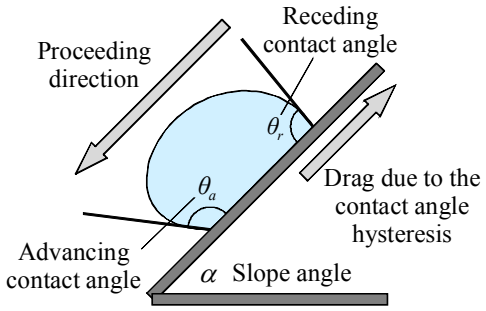


Figure 2. Advancing and receding contact angles of a water droplet.

With regard to a droplet which lies on a solid plane, the capillary forces along the circular triple boundary cancel each other out if the contact angle is constant along the circle. When some external forces are put on the droplet and its shape is deformed, the contact angles vary while the droplet stands still until the contact angle variance reaches at a certain value.

The contact angle hysteresis is defined as the difference between the advancing and receding contact angles (θ_a and θ_r, respectively). These two angles are defined as the largest and the smallest contact angles, respectively, at the moment that the water droplet starts moving on the solid plane by the sufficient external force. The slope angle at this moment is called the falling angle. Figure 2 illustrates the advancing and receding contact angles for an incline.

While the droplet is moving on the plane, a drag operates on the droplet toward the reversed direction against the proceeding direction. The amount of drag is related to the contact angle hysteresis. Assuming that the shape of the triple

boundary is a circle, the drag F_{hys} is approximated with the following equation [Car95a]

$$F_{hys} \approx \frac{1}{2}\pi r\gamma_L(\cos\theta_r - \cos\theta_a), \quad (2)$$

where, r represents the radius of the water droplet. θ_r and θ_a are the receding and the advancing contact angles, respectively.

3.3 Wind Drag

Automobile windshields meet with air resistance, or wind drag, according to the velocity of the running vehicle. The wind drag is defined as follows:

$$F_{wind} = \frac{1}{2}\rho C_D S V^2, \quad (3)$$

where, ρ is the density of the air, C_D is the coefficient of resistance, S is the projected size of the droplet, and V is the velocity relative to the air.

In Equation (3), the droplet is assumed to be floating in the air. Since all droplets in our model are placed on a solid windshield, the equation needs to be modified. We assume that the wind is weakened at places very close to the solid plane. It is known that in such near-boundary layer, the wind velocity changes in a complicated manner.

We employed a simplest compensation to decrease the velocity in the near-boundary layer using an exponential law as shown in the following formula.

$$\tilde{V} = \begin{cases} V\left(\frac{y}{\delta}\right)^{\frac{1}{2}} & (y < \delta) \\ V & (y \geq \delta), \end{cases} \quad (4)$$

where, V is the wind velocity out of the boundary layer (relative to the solid plane), y is the height of the droplet, δ is a parameter representing the thickness of the boundary layer, and $\tilde{V}$ is the compensated wind velocity for the droplet.

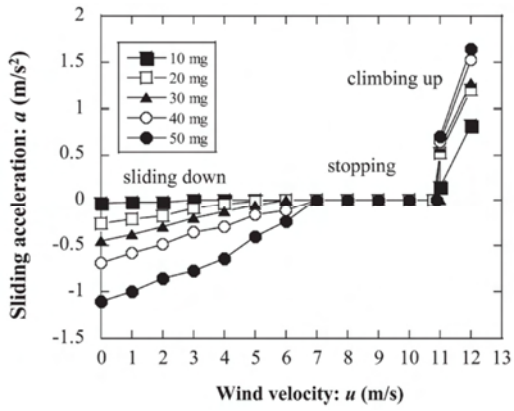


Figure 3. A measured relationship between the wind velocity and the acceleration of droplets, using a varying droplet size as a parameter (excerpt from [Has08a]).

3.4 Viscous Dissipation

When a droplet is moving or rolling, another drag is caused by some in-bulk friction called viscous dissipation [Bic05a]. The drag is in proportion to the velocity of the droplet and represented as

$$F_{bulk} = \eta R v f(\theta), \quad (5)$$

where, η is the degree of viscosity of the water, R is the radius of the droplet, v is the velocity of the droplet. $f(\theta)$ is a factor dependent on the contact angle.

3.5 Wind Speed and the Droplet Acceleration

In the surface finishing engineering discipline, Hashimoto et al. [Has08a] introduced an experiment to measure the acceleration of various volumes of water droplets placed on an angled hydrophobic plane in a wind tunnel. Figure 3 quotes from the literature and shows the result of the measured descending or ascending acceleration of the droplets. The contact angle, the slope angle, and the falling angle are 105° , 35° and 10°, respectively.

In the range where the wind velocity is relatively low, moderate but more falling accelerations are observed as the droplet size becomes greater. When the wind velocity is raised beyond a certain value (7m/s in Figure 3), the droplet stays still within some range of wind velocities. When the velocity is further raised beyond a higher value (11m/s), rapid ascending accelerations are observed, which are greater as the droplet becomes larger.

On the other hand, we simulated the sliding accelerations of a droplet taking the following five forces into account (Figure 4).

- Gravity (vertical) F_g
- Wind drag (horizontal) F_{wind}
- Perpendicular force (normal to windshield)

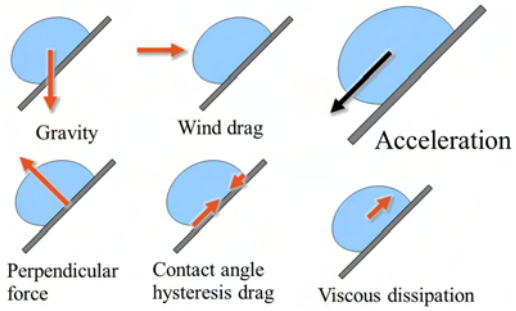


Figure 4. External forces added to a droplet and the resultant acceleration. In this example, the gravity is more dominant than the wind drag and thus the droplet slides down.

- Viscous dissipation drag (tangential to windshield) F_{bulk}
- Contact angle hysteresis drag (tangential to windshield) F_{hys}

The wind drag F_{wind} has been described in Section 3.3. The contact angle hysteresis drag behaves as a resistance force parallel to the windshield, in the same way as the perpendicular force normal to the windshield. The force F_{hys} represented in Equation (2) defines the maximum limit of the hysteresis drag.

In our implementation, the maximum limit is specified by a dimensionless coefficient $\Omega \equiv \cos\theta_r - \cos\theta_a$. Since the relationship between the wind velocity and the contact angles is hard to simulate, we approximate the Ω value as a function of the wind velocity V. When the velocity is small, we force the Ω value to keep a minimum constant Ω_{min} which is typically 0.5.

$$\Omega(V) = \max\left[\Omega_{min},\ 2\left\{1 - \exp\left(-\frac{V}{\sigma}\right)\right\}\right], \quad (6)$$

where, σ is a constant parameter which controls the saturation rate of $\Omega(V)$. When the wind is extremely strong, the contact angles are assumed to be also as extreme as $\theta_a \to 180°$, $\theta_r \to 0°$, and thus $\Omega(\infty) \to 2$. This is well accounted for by Equation (6).

Figure 5 shows a simulated result of the accelerations for the varying droplet sizes. The range of wind velocities in which the droplet stays still is reproduced, and the range is very similar to the measured result in Figure 3.

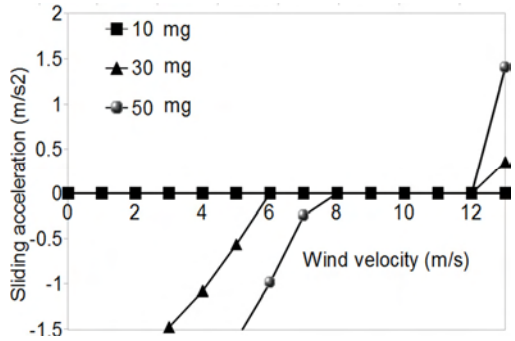


Figure 5. Simulation results of the droplet accelerations.

3.6 Collision between Droplets

The surface tension of the water droplet causes a pressure difference in the droplet. This is known as the Laplace pressure and is greater as the droplet radius is smaller. Therefore, when two water droplets of different sizes collide with each other, the small droplet gets absorbed by the larger one.

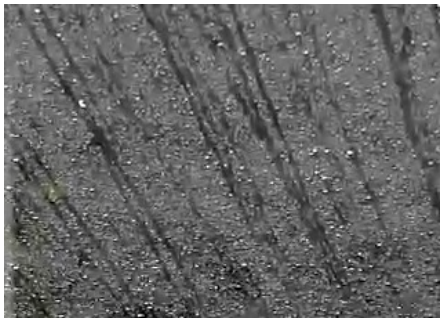


Figure 6. Droplet trajectories caused by the Lotus effect (image captured from a live-action movie of a windshield).

We implemented this process and it is invoked on droplet collision detection.

3.7 Distribution of Raindrop Radii and the Lotus Effect

Lotus effect is a phenomenon which occurs when a water droplet moves across a hydrophobic surface. Lots of very small droplets and contamination spread on the surface are removed by the moving droplet along the trajectory. The same phenomenon is observed on a windshield as demonstrated in the snapshot of Figure 6.

Figure 7, an excerpt from [Fur02a], is a rain droplet radius distribution under 1mm/h rainfall. The graph is with the raindrop diameters as the horizontal axis and the number of raindrops for each diameter as the vertical logarithmic axis. The line indicated as 'MP' is an exponential distribution model called the Marshall-Palmer distribution [Mar48]. Each graph legend is the place name of the observing site. Some legends contain observing periods in months.

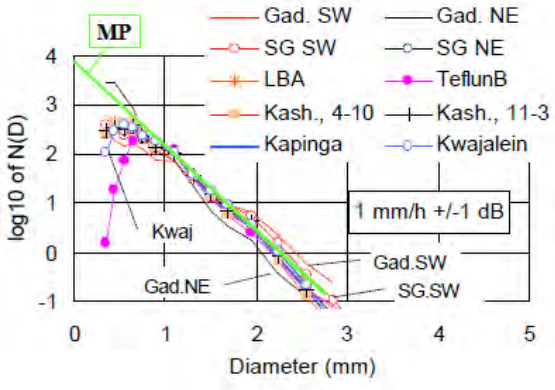


Figure 7. Distribution of the number of raindrops for each diameter (drop size distribution). Each graph legend indicates the name of the observing site (excerpt from [Fur02a]).

According to the model, the smaller the raindrop diameter is, the greater the number of raindrops becomes. Especially, tiny raindrops of below 1mm are contained with an exponentially large numbers. Therefore, it is impractical to simulate the motion of every droplet. Fortunately, those tiny raindrops do not move at all with our simulation model as shown in Figure 5. Thus we apply a single large normal map onto the windshield. The map contains the normal vectors which represents all the small droplets standing still on the windshield.

4. IMPLEMENTATION AND RESULTS

This section describes implementation of our method proposed in the previous section and demonstrates some results.

4.1 Implementation Overview

We implemented the system on top of Unity, a popular game engine. Although our method regards each water droplet as a particle, we implemented each droplet as a small rigid body which does not rotate. Regarding the rigid body physics engine, we used NVIDIA PHYSX embedded in the Unity system.

The flow of the whole process is outlined as follows.

- Initialization
- Main loop
  - Droplet generations
  - Physics simulation
  - Collision detection
    - Droplet mergers
    - Droplet deletions
  - Updates of large droplet shapes
  - Update of windshield alpha map (Lotus effect over small droplets)
  - Rendering

4.2 Physics Simulation of Droplets

In each time step of the simulation, our system calculates the external forces imposing on the water droplets as illustrated in Figure 4.

Regarding the gravity, we added some random noise to the force component parallel to the windshield in order to realize natural motions of the droplets caused by some assumed fluctuation of the running vehicle.

The implementation of viscous dissipation (Section 3.4) is a heuristic matter since the factor $f(\theta)$ in Equation (5) is not determined. We used a constant value $\eta f(\theta) = 0.5$ in the equation. The important point is that the viscous dissipation drag F_{bulk} is in proportion to the droplet velocity. The above constant value can be used to control the maximum droplet speed.

While the droplets are moved by the external forces, we obtain each collision point with its u-v coordinates and the normal vectors of the colliders from the collision detector of the physics engine. For a droplet being regarded as to be on the windshield, the windshield point corresponding to the droplet is calculated and the refraction map image for the Lotus effect is updated.

In case that a droplet collides with another droplet, the Laplace pressure effect is applied. The system compares the masses of the two droplets. If the difference is greater than the pre-defined threshold, these two will fuse together into one droplet.

4.3 Rendering Large, Movable Droplets

Each large water droplet (with over 1mm diameter) is rendered as a disc-shape polygon mesh when it is staying still on the windshield. The normal vectors on the disc surface are controlled so that the refracted environment appears to be mapped on a hemisphere.

While the droplet is moving across the windshield, its shape is deformed to be longer along the moving direction. The normal vectors are controlled so that the lengthened transparent droplet looks like a drug capsule sectioned by a screen-parallel plane. The deformation is controlled so that the assumed volume of the droplet is preserved. Using its normal vectors, the pixel shader calculates the refraction directions and maps the background texture image as the environment. Figure 8 is a close-up rendering image of a pseudo-hemisphere water droplet and a deformed pseudo-hemisphere.

Those large droplets are generated with various

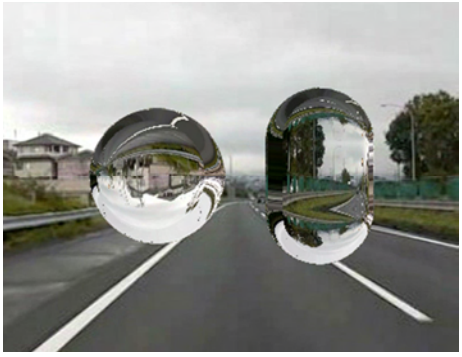


Figure 8. Droplets rendered as a pseudo-hemisphere (left) and a deformed pseudo-hemisphere (right).

sizes according to the Marshall-Palmar distribution shown in Figure 7. The number of large droplets generated per frame is set to be five typically. They are accumulated but eventually moved away out of the windshield or collided and fused with others. As a result, a couple of hundred to one thousand large droplets reside in the steady-state situation.

4.4 Rendering Small and Still Droplets

Small droplets (with less than 1mm diameter) are represented as perturbation in a normal map image for the windshield, as described in Section 3.7. The diameters of the generated small droplets vary also according to the Marshall-Palmar distribution. The number of small droplets in our implementation amounts to approximately 800K$/m^3$.

The outside scene image is refracted according to the normal map. The trajectories of large droplets (pseudo-hemispheres) are stored as an image component which is used to suppress the normal map. They are composed in the shader program and the Lotus effect on the windshield surface is rendered (Figure 9).

Figure 9. The Lotus effect. Small and still droplets are rendered as a normal map on the windshield. Large and moving droplets are rendered as pseudo-hemispheres.

4.5 Performance

All results referred to in this section are captured snapshots of real-time animations rendered from the driver's point of view toward the automobile proceeding direction viewing the outside through the windshield. The source of the outside image is a motion picture shot with a video camera placed between the two front seats of a running car when no rain is falling. The pre-recorded image is mapped as a video texture onto a billboard model placed in front of the windshield model.

Figures 10 and 11 are the examples with a small wind velocity. In Figure 10, a relatively large contact angle hysteresis is specified and thus the adherence is strong that the droplets do not move at all. In Figure 11, the adherence is smaller and the droplets move along the windshield curve.

Figure 10. A result with low wind velocity (11.3m/s) and a large contact angle hysteresis with $\Omega_{min} = 0.5$.

Figure 11: A result with low wind velocity (11.3m/s) and a small contact angle hysteresis with $\Omega_{min} = 0.05$.

Figure 12. A result with high wind velocity (15m/s) and a large contact angle hysteresis with $\Omega_{min} = 0.5$.

Figure 12 is a result with stronger wind and the large droplets climb straight up the windshield. Since the adherence is strong and the boundary layer is set to be thick, the small droplets are made still.

The frame rates for Figures 10, 11 and 12 are 134-153fps, 80-100fps, and 70-100fps, respectively. The scene contains a windshield, large droplets and the video texture billboard shapes, which total approximately 17K vertices.

4.6 Rendering Conditions

For the rendering results, we used an Intel Core2 Extreme X9600 (3GHz), NVIDIA GeForce GTX480 Graphics and 8GB main memory. The horizontal field of view was 45° and the distance between the viewpoint and the windshield was approximately 0.5m. The horizontal curvature radius of the windshield geometry was 5m constant and the vertical curvature was 0 (flat). The slope of the windshield was inclined at a 45° angle.

5. CONCLUSION AND FUTURE WORK

We proposed a real-time animation method which reproduces the behaviour of a group of water droplets on a hydrophobic windshield. We modeled each of large droplets as a mass point and took into account dynamic hydrophobicity by employing the contact angle hysteresis which causes appropriate adherence for each droplet.

We also compared the accelerations of simulated droplets with those of measured real water droplets from literature of surface finishing engineering analysis. By introducing a near boundary layer where the wind is reasonably weakened, our result matched the measured one and reproduced realistic behaviours of the droplets.

For a huge number of tiny water droplets which do not move in our model, we introduced a normal map applied to the windshield. By using the image-based droplets, the Lotus effect was effectively reproduced.

For practical number of large droplets, our method runs in real-time and can be easily adopted as an effect for video games and vehicle simulators. The performance is degraded when the large droplets are not blown off and accumulated on the windshield because the motion simulation is done on a per large droplet basis.

Future work includes the performance improvement for larger number of droplets, more realistic deformation of the droplets, and handling of uneven wind velocity distributions.

REFERENCES

[Bic05a] Bico, J., Basselievre, F., and Fermigier, M. Windswept droplets. Bulletin of the American Physical Society 2005, 58th Annual Meeting of the Division of Fluid Dynamics, 2005.

[Car95a] Carre, A., and Shanahan, M.E.R. Drop motion on an inclined plane and evaluation of hydrophobia treatments to glass. Journal of Adhesion, Vol.49, No.3-4, pp.177-185, 1995.

[Fou98a] Fournier, P., Habibi, A., Poulin, P., Simulating the flow of liquid droplets. Graphics Interface, pp.133-142, 1998.

[Fur02a] Furutsu, T., Shimomai, T., Reddy, K.K., Mori, S., Jain, A. R., Ong, J.T., Wilson, C.L. Comparison of the characteristics of the drop size distributions in the tropical zone (In Japanese). Open Workshop 2002 on Coupling Processes in the Equatorial Atmosphere, 2002.

[Has08a] Hashimoto, A., Sakai, M., SONG, J.-H., Yoshida, N., Suzuki, S., Kameshima Y., and Nakajima, A. Direct observation of water droplet motion on a hydrophobic self-assembled monolayer surface under airflow. Journal of Surface Finishing Society of Japan, Vol.59, No.12, pp.907-912, 2008.

[Hir08a] Hirvi, J.T., and Pakkanen, T.A. Nanodroplet impact and sliding on structured polymer surfaces. Surface Science. No.602, pp.1810–1818, 2008.

[Kan93a] Kaneda, K., Kagawa, T. and Yamashita, H. Animation of water droplets on a glass plate. Proc. Computer Animation '93, pp.177–189, 1993.

[Kan96a] Kaneda, K., Zuyama, Y., Yamashita, H. and Nishita, T. Animation of water droplet flow on curved surfaces. Proc. Pacific Graphics '96, pp.50–65, 1996.

[Kan99a] Kaneda, K., Ikeda, S., and Yamashita, H. Animation of water droplets moving down a surface. Journal of Visualization and Computer Animation, Vol.10, No.1, pp.15-16, 1999.

[Kor97a] Korlie, M. S., Particle modeling of liquid drop formation on a solid surface in 3-D. Computers & Math. with Applications, Vol.33, No.9, pp.97-114, 1997.

[Mar48a] Marshall, J.S., and Palmar, W.McK. The distribution of raindrops with size. Journal of Meteorology, Vol.5, pp.165-166, 1948.

[Ric99a] Richard, D., and Quere, D. Viscous drops rolling on a tilted non-wettable solid. Europhys. Lett., Vol.48, No.3, pp.286-291, 1999.

[Sak06a] M. Sakai, J-H Song, N. Yoshida, S. Suzuki, Y. Kameshima and A. Nakajima, Relationship between sliding acceleration of water droplets and dynamic contact angles on hydrophobic surfaces. Surface Science, 600 (16), 204-208, 2006.

[Suz09a] Suzuki, S., Nakajima, A., Sakurada, Y., Sakai, M., Yoshida, N., Hashimoto, A., Kameshima, Y., Okada, K. Mass Dependence of rolling/slipping ratio in sliding acceleration of water droplets on a smooth fluoroalkylsilance

coating. Europhys. Lett., Vol.48, No.3, pp.286-291, 2009.

[Tat06a] Tatarchuk, N. Artist-directable real-time rain rendering in city environments. Course Note #26, SIGGRAPH 2006.

[Thu10a] Thürey, N., Wojtan, C., Gross, M., and Turk, G. A multiscale approach to mesh-based surface tension flows. ACM TOG, Vol.29, No.4 (Proc. SIGGRAPH 2010), 48, 2010.

[Wan05a] Wang, H., Mucha, P.J., and Turk, G. Water drops on surfaces. ACM TOG, Vol.24, No.3 (Proc. SIGGRAPH 2005), pp.921-929, 2005.

[Zha11a] Zhang, Y., Wang, H., Wang, S., Tong Y., and Zhou, K. A deformable surface model for real-time water drop animation. IEEE TVCG, PrePrint, August 2011.

Visualization of Very Large 3D Volumes on Mobile Devices and WebGL

José M. Noguera, Juan-Roberto Jiménez
Graphics and Geomatics Group of Jaén. University of Jaén.
Campus Las Lagunillas, Edificio A3, 23071 Jaén, Spain.
{jnoguera,rjimenez}@ujaen.es

ABSTRACT

Platforms based on OpenGL ES 2.0 such as mobile devices and WebGL have recently being used to render 3D volumetric models. However, the texture storage limitations of these platforms cause that only low-resolution models can be visualized. This paper describes a novel technique that overcomes these limitations and allows us to render detailed high resolution volumes on these platforms. Additionally, we propose a software architecture that permits existing volume rendering techniques to be adapted to mobile devices and WebGL. A set of experiments has been carried out to assess the performance of the proposed architecture on these platforms with different volumes of increasing resolution. Results prove that our proposal is feasible, robust and achieves visualization of very large volumes on constrained platforms.

Keywords: Volume visualization, OpenGL ES, mobile devices, WebGL, large volumetric models, software architecture.

1 INTRODUCTION

Nowadays mobile devices are extensively used as a worthy tool in many different scenarios of our life. Their hardware and software capabilities are constantly being enhanced, and recent research has demonstrated their validity to compute complex computer graphics algorithms. In fact, it has been proved that the volume visualization field can benefit from the properties of mobile devices in many interesting applications [18, 1, 5, 7, 9, 11, 13, 22].

However, it is a common misunderstanding to assume that the same results can be achieved by a literal translation to mobile devices of classic algorithms originally developed for standard PCs or workstations. There are two important factors that must be taken into account:

- The standard graphics specification of this kind of devices is OpenGL ES 2.0 [12], which differs from the desktop PC counterpart in several aspects, e.g., the lack of 3D texture support.
- These devices must rely on batteries, so their hardware and software architectures are designed to favour power-efficiency instead of pure computing power.

In addition, recent advances in display technologies allow today's mobile devices to feature large and high resolution screens, which require large volumetric models in order to achieve a minimum of quality. For example, newer tablets such as the iPad3 feature screen resolutions that surpass the Full-HD standard used in most monitors and TV screens. Nevertheless, their GPU and memory capacities are still limited and do not support large models directly.

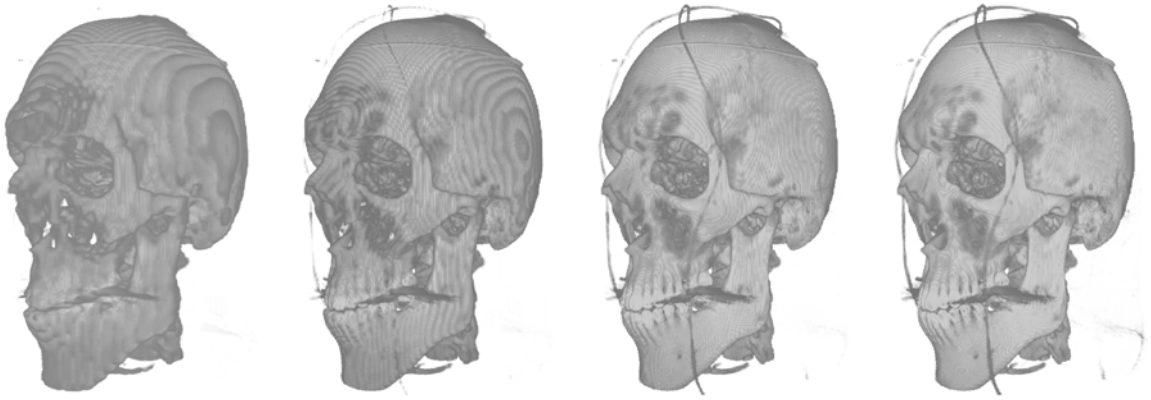
In this paper, we deeply study this problem in the context of direct volume rendering. We present a proposal to render very large volumetric models in order to meet the user expectations in quality and performance by overcoming the referred limitations of handheld devices, see Figure 1. Moreover, we describe a software architecture that allows us to adapt existing volume rendering techniques based on 3D textures to platforms that only support 2D textures.

The ideas described in this paper are also applied to WebGL[1], the standard for accelerated graphics on the Web. As this standard is based on the same specification used by mobile devices, i.e. OpenGL ES 2.0, it suffers from the same limitations, including the lack of 3D textures.

Finally, we have implemented a mobile and a WebGL based prototypes and conducted a set of experiments to test performance of these platforms under the conditions of maximum storage requirements.

The rest of the paper is organized as follows. Section 2 presents current research in volume visualization techniques for mobile devices and WebGL. In Section 3,

[1] https://www.khronos.org/registry/webgl/



64×64×256 128×128×256 256×256×256 512×512×256

Figure 1: Volumes of increasing resolution. Images rendered using the proposed technique.

our novel technique and architecture for volume rendering are presented. Section 4 presents our performance evaluation and discusses the results. Finally, Section 5 concludes the paper.

2 PREVIOUS WORK

In the context of scientific visualization and volume rendering, a volume is usually represented as a set of images/slices that are parallel and evenly distributed across the volume. Equation 1 expresses the computation of the final color of a given pixel by composing the colors and opacities of the samples along a given line across the volume, for a certain wavelength λ [10]:

$$C_\lambda(x) = \sum_{k=0}^{n} c_\lambda(x+r_k)\alpha(x+r_k) \prod_{l=k+1}^{n} (1-\alpha(x+r_l)) \quad (1)$$

where $C_\lambda(x)$ is the final color at a given position x, $c(x+r_k)$ is the color of the kth sample at position $x+r_k$ inside the volume and $a(x+r_k)$ is its corresponding opacity.

Volume visualization algorithms have not been applied to mobile devices until recently. First attempts overcame the mobile devices limitations by employing a server-based rendering approach. This approach relies on a dedicated rendering server that carries out the rendering of the volume and streams the resulting images to the mobile client over a network [7, 9]. Also following a server-client scheme, Zhou et al. [22] employed a remote server to precompute a compressed iso-surface, which is sent to the mobile device allowing a faster rendering. Moser and Weiskopf [11] introduced an interactive technique for volume rendering on mobile devices that adopts the 2D texture slicing approach. Noguera et al. [13] proposed an algorithm that overcomes the 3D texture limitation of mobile devices and achieves interactive frame rates by caching the geometry of the slices in a vertex buffer object (VBO). ImageVis3D [5] is an iOS application that uses the 2D texture slicing approach. While the user is interacting the number of slices is drastically reduced. At the end of an interaction a new image is rendered with the whole set of slices. This rendering step is carried out in the mobile device itself, or in a remote server in case of complex or large models. Focused on the visualization of bones, Campoalegre [18] also proposed a client-server scheme where the model is compressed in the server side by the Haar Wavelet function and reconstructed in the client device. On the other hand, Congote et al. [1] implemented a ray-based technique using the WebGL standard.

All the aforementioned techniques share the same limitation: the lack of 3D textures on OpenGL ES 2.0 and WebGL severely restricts the size and resolution of the volumetric models that can be rendered. The problem is aggravated on WebGL, as these applications are usually run on desktop computers equipped with large monitors.

The following proposals deal with the problem of very large volumetric models but in the context of PC or workstations whose features differ from the intrinsic peculiarities of mobile devices and WebGL. A straightforward method to deal with a large volume is the bricking technique [2]. This technique subdivides the volume into several smaller blocks in such a way that a single block fits into texture memory. Gunthe and Straßer [3] used a wavelet based volume compression in order to render large volume data at interactive frame rates in a standard PC. Tomandl et al. [17] combined local and remote 3D visualization (standard PC + high-end graphics workstation) achieving low-cost but high-quality 3D visualization of volumetric data. Schneider and Westermann [15] also overcame the problem of the limited texture memory by compressing large scale volumetric data sets. Their solution takes advantage of temporal coherence on animated environments. Thelen et al. [16] introduced a dynamic subdivision scheme incorporating multi-resolution wavelet representation to

visualize data sets with several gigabytes of voxel data interactively on distributed rendering clusters. Finally, Xie et al. [21] subdivided the volume dataset into a set of uniform sized blocks and combined early ray termination, empty-space skipping and visibility culling techniques to accelerate the rendering process.

3 METHODOLOGY

This section details the algorithm, the software architecture and the implementation details that we propose to render large volumetric models on handheld devices and WebGL. Volumes are usually stored as a set of slices, each one containing a 2D image that represents the intersection of the volume with the slice. Common volume rendering approaches [19] store these slices in a 3D texture. However, neither mobile devices nor WebGL support 3D textures. This limitation can be overcome by storing the slices in a single 2D texture following a mosaic configuration [1, 13]. Nonetheless, without recurring to external servers, this technique limits the size of the volume that can be stored because 2D textures are considerably smaller than 3D textures.

We extend this mosaic configuration solution to exploit the maximum texture capacity of the GPU in order to deal with larger volumetric models. Our idea is based on maximizing the multi-texture storage capacity of the device by using all the available texture units and color channels. Usually, current handheld devices are able to store up to 8 RGBA textures of 2048^2 texels each. These numbers give us a maximum volume size of 512^3 voxels when using our technique, which is considerably larger than the models rendered until now on mobile platforms.

Our technique stores the 3D volume by placing each slice one next to the other in a given color channel of a 2D texture. If the texture dimensions are exceeded, we continue storing slices in the next color channel. This way, data-level parallelism is optimized [6, 20]. When all the channels of the texture are completed, the remaining slices are stored in consecutive texture units following the same pattern. Figure 2 shows an example of a 2D texture where each color channel stores a subset of slices in a mosaic configuration. Thus, each RGBA color represents the values inside four different non-consecutive slices of the volume.

Our storage configuration technique can be utilized with any standard volume rendering approach based on 3D textures. Figure 3 illustrates our proposal for a volume rendering architecture designed for OpenGL ES 2.0. This architecture is divided into two main parts: the *texture memory* and the *shader*. The texture memory is used to store both the 3D volume and the transfer function. The volume is stored using multi-textures as previously described. On the other hand, the transfer function refers to the texture normally required

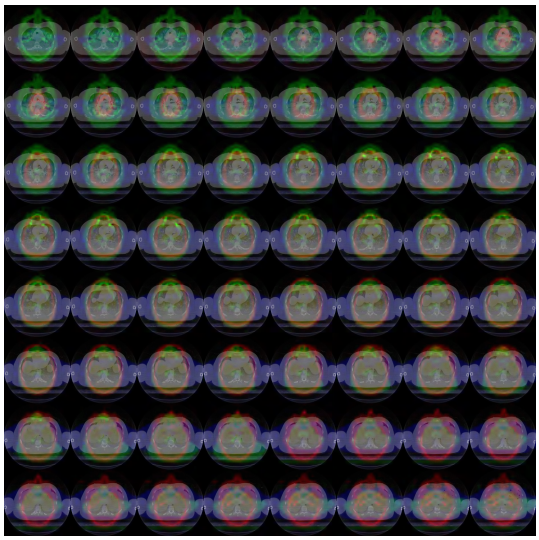


Figure 2: Four mosaics stored in an RGBA texture.

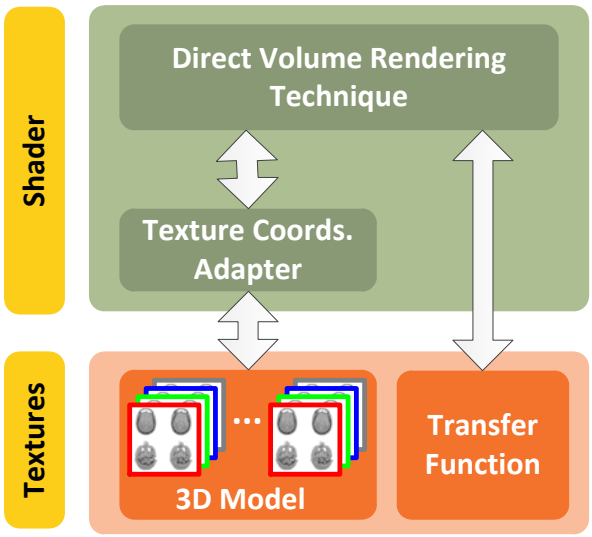


Figure 3: Software architecture of our proposal.

by volume rendering techniques to assign a color to each voxel [19]. Thereby, different parts of the model (bones, muscles, etc.) can selectively be emphasized by interactively modifying the transfer function.

In the shader section there are two modules: the *texture coordinate adapter* (TCA) and the selected *direct volume rendering* (DVR) technique. The DVR technique sends a 3D coordinate to the TCA and receives back an interpolated grey-tone. This tone is then converted into an RGBA value according to the given transfer function and used to compute the color of the corresponding fragment. It is important to remark that the DVR technique is unaware of the underlying 3D model storage method. In fact, this architecture provides an straightforward mechanism to adapt existing DVR techniques to the platforms we are interested in this paper.

$$
\begin{aligned}
aux_1 &= floor(S * z) \\
aux_2 &= floor\left(\frac{aux_1}{M_x M_y}\right) \\
depth &= min\left(1.0, \frac{M_x M_y}{S}\right) \\
z_{ini} &= aux_2 * depth \\
z_{res} &= \frac{z - z_{ini}}{depth} \\
u_1 &= floor\left(\frac{aux_2}{Max_{ch}}\right) \\
ch_1 &= mod(aux_2, Max_{ch})
\end{aligned}
$$

Listing 1: Computation of the texture unit u_1, the color channel ch_1 and the corresponding (x, y, z_{res}) from the 3D texture coordinate (x, y, z).

The TCA module transforms the 3D texture coordinate provided by the DVR technique to a format suitable for the volume representation and computes the grey-tone. The returned value includes the trilinear interpolation with the one-voxel distance neighbourhood. The process performed by this module can be decomposed into two differentiate tasks.

The first task consists of deriving the texture unit u_1, the RGBA channel ch_1 and the new local 3D texture coordinates (x, y, z_{res}) from the original 3D texture coordinates (x, y, z). Listing 1 shows how to compute these values. S is the number of slices in the volume, $M_x \times M_y$ is the maximum number of slices in a mosaic stored in a single color channel of a texture, see Figure 2, and Max_{ch} is the number of channels per texture. As the slices are stored in a consecutive manner along the channels and texture units, each mosaic stores an interval of the z-component of the full volume. The value z_{res} is the residual z defined as the original z minus the z coordinate of the voxel stored in the first texel of the selected mosaic.

The second task is devoted to compute a pair of 4-tuples (u_1, ch_1, x_1, y_1), (u_2, ch_2, x_2, y_2) that define two texels from the set of 2D textures, where u_i, ch_i refers to the texture unit and the color channel, respectively, and x_i, y_i are the 2D texture coordinates of the desired texel in the corresponding mosaic stored in the texture unit u_i. These two texels are neighbours along the z direction, and are placed in the same or in consecutive mosaics. The grey-tones of these texels are merged to simulate trilinear interpolation. Note that bilinear interpolation is automatically obtained by the texture interpolator of the GPU. Listing 2 shows how to perform these operations. Here, T is an array of 2D samplers that contains the volumetric model.

$$
\begin{aligned}
S_{tex} &= min(M_x M_y, S) \\
aux_3 &= floor(z_{res} * S_{tex}) \\
aux_4 &= mod(aux_3 + 1, S_{tex}) \\
x_1 &= \frac{aux_3}{M_x} + \frac{x}{M_x} \\
y_1 &= \frac{floor\left(\frac{aux_3}{M_y}\right)}{M_y} + \frac{y}{M_y} \\
x_2 &= fract\left(\frac{aux_4}{M_x}\right) + \frac{x}{M_x} \\
y_2 &= \frac{floor\left(\frac{aux_4}{M_y}\right)}{M_y} + \frac{y}{M_y} \\
next &= c + step(aux_3, aux_4) \\
u_2 &= t + step(Max_{ch}, next) \\
ch_2 &= mod(next, Max_{ch}) \\
v_1 &= tex2D(\ T[u_1],\ (x_1,\ y_1)\ )[ch_1] \\
v_2 &= tex2D(\ T[u_2],\ (x_2,\ y_2)\ )[ch_2] \\
V &= mix(v_1,\ v_2,\ z_{res} * S_{tex} - aux_3)
\end{aligned}
$$

Listing 2: Computation of V, the value of the grey-tone at position (x, y, z).

The shaders represented by Listings 1 and 2 have carefully been designed in order to avoid flow control operators, when possible, by promoting the use of built-in GLSL functions like *step*. Observe that the use of flow control operators have a cost in the GPU of mobile devices.

4 RESULTS

In order to measure the effectiveness and performance of our technique two prototypes have been implemented. The first one is a mobile application using OpenGL ES 2.0 and the second one is a desktop solution using WebGL. The selected technique for the DVR module is a ray-based technique implemented in the GPU [4, 8]. This technique basically consists of a loop of n steps that traverses the volume accumulating color and opacities along a given ray-direction.

Recall that our architecture is independent of the visualization technique, and a faster texture-based approach could have been used instead [13]. However, our goal was not to measure the performance of the DVR technique, but to assess the performance and scalability when the resolution of the 3D model increases and multiple textures are used.

In our experiments, we used the *CT human* male dataset provided by the *Visible Human Project*[2]. This dataset has a total resolution of $512^2 \times 1877$ voxels.

4.1 Results on Mobile Devices

The experiments were conducted on an iPad2 tablet. This device features a dual core PowerVR SGX543MP2 GPU and the iOS 5 operating system. The test application was developed as a native iOS application, using C++ and GLSL ES 2.0. According to Apple's technical specifications, this device supports a maximum 2D texture size of 2048^2 texels, and up to 8 texture units.

Our experiments intended to cover all the range of model resolutions provided by our technique. We used a subset of the CT human dataset, shown in Figure 4, with different resolutions. For each experiment, a 100 frame animation of the camera rotating around the CT human model was generated, and the mean times needed to render each frame were taken. Due to the tile-based deferred rendering architecture [14] used by the GPU, OpenGL ES calls can be deferred until the scene is presented. In order to perform exact timings, we forced the frame rendering to finish by means of a *glFinish* call. Figure 5 shows graphically the results obtained in milliseconds. The results for the following experiments are included:

- 128^3A: stored using one single-channel texture.
- 128^3B: the same model as the previous experiment.
- 256^3A: stored using one RGBA texture.
- 256^3B: stored using four single-channel textures.
- $512^2 \times 384$: stored using six RGBA textures.

The experiment 128^3A utilized a simplified version of the TCA module that only handles one mosaic, similar to the proposal of Congote et al. [1], while the experiments 128^3B, 256^3A, 256^3B and $512^2 \times 384$ used our proposed TCA module that can deal with high resolution 3D models.

Note that $512^2 \times 384$ is the maximum resolution that can be achieved by our technique on this device, because one texture unit is used by the transfer function and another one is required by the rendering technique.

In all cases, uncompressed 2048^2 textures were used. The ray-based DVR technique performed 80 steps in all the experiments. The screen resolution was 480×320 pixels.

Results in Figure 5 show that the rendering times are almost constant among all the tested datasets when using

[2] `http://www.nlm.nih.gov/research/visible/visible_human.html`

Figure 4: The CT human model on an iOS mobile device. Resolution 256^3 using 4 textures and 80 steps-raytracing.

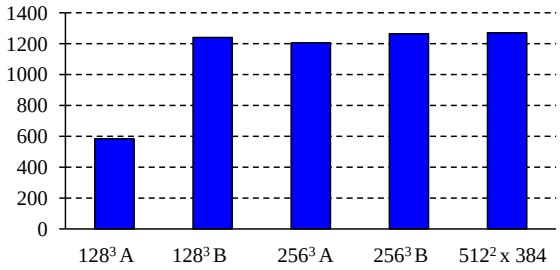


Figure 5: Rendering time (ms) for an iPad2 mobile device. Screen resolution: 320×480 pixels. Raytracing: 80 steps.

the same TCA module. This suggests that the resolution of the volumetric model does not have a significative impact on the performance of the rendering process, as long as the model fits in the device's memory. The performance of the volumetric rendering depends on the screen resolution and on the number of steps performed during the raycasting rather than on the model resolution, as it was studied in [1, 13].

As stated above, the experiments 128^3A and 128^3B were performed using the same dataset but different TCA modules. The experiment 128^3B used our technique to compute the texture coordinates whereas 128^3A used a simplified one. It was possible to use this simplified version because the model is small enough to fit in one mosaic. As shown in Figure 5, the rendering time of the second experiment doubles the time achieved by the first one. Our proposed technique neither increases the number of texture accesses nor

adds additional conditional branches in the shader. Nevertheless, it increases its longitude by about a dozen of straightforward code lines to handle additional color channels and texture units, see Listings 1 and 2. Albeit these lines only perform simple computations, they are repeated once per step.

The experiments 256^3A and 256^3B employed a dataset that is too large for one mosaic of 2048^2 pixels. Two strategies can be used to store it: we can use either one RGBA texture (256^3A) or four single-channel textures (256^3B). Our experiments show that there are no significative differences between both strategies in terms of rendering times, and as a consequence, we can use the best suited for our particular application.

Finally, we also prove that our technique allows us to exploit all the available texture resources of the mobile device in order to render a very large dataset. We manage to render a model of up to $512^2 \times 384$ voxels while keeping the same rendering speed.

4.2 Results on WebGL

Following, we conducted a similar set of experiments to test the performance of the WebGL implementation. The tests were carried out using an Intel Core2 Quad CPU Q6600, 4 GB of RAM, a GeForce 8800GT and Windows 7 SP1 64 bits. As web browser, we tested Opera 11.50 labs (build 24661).

The selected GPU supports 2D texture sizes of up to 8192^2 texels and provides 16 texture units. Given that the texture size is considerably larger than the provided by the iPad2, we used a larger subset of the CT human dataset for our experiments, as shown in Figure 6.

In order to measure the rendering times, we forced the WebGL canvas to redrawn continuously and counted the number of frames rendered during an animation. This animation consisted of the camera rotating around the model during 5 seconds. Figure 7 shows graphically the mean times needed to render each frame in milliseconds. The results for the following experiments are included:

- $512^2 \times 256$A: stored using one single-channel texture.
- $512^2 \times 256$B: the same model as the previous experiment.
- $512^2 \times 1024$: stored using one RGBA texture.

The experiment $512^2 \times 256$A utilized the simplified version of the TCA module described in Section 4.1, while the experiments $512^2 \times 256$B and $512^2 \times 1024$ used our proposed TCA module. In this case, we used uncompressed 8192^2 textures and 128 steps for the ray-casting. The WebGL canvas resolution was 800^2 pixels.

We found that Opera stopped working every time we tried to load more than one 8192^2 texture. The reason was that the NVIDIA driver exceeded the Windows imposed rendering time limit (TDR) of two seconds. This limited our experimentation to a model of $512^2 \times 1024$ voxels, which is the maximum size that can be encoded using all color channels of a single 8192^2 RGBA texture.

Interestingly, we did not run into these problems when we were experimenting with mobile devices, in spite of the fact that both platforms share the OpenGL ES 2.0 specification. In our opinion, today's WebGL implementations are still relatively immature and the tested mobile device proved to be a more predictable and stable platform. As opposed to WebGL under Opera, the iPad2 was able to correctly handle all our experiments, including those using the maximum texture size on all the available texture units.

Nevertheless, comparing Figures 5 and 7 we can easily observe that WebGL is more than one order of magnitude faster than the selected mobile device when rendering a similar volumetric model, even with a commodity desktop PC.

The experiments $512^2 \times 256$A and $512^2 \times 256$B (see Figure 7) used a model that can be stored in a single mosaic. Therefore, our proposed TCA module was not strictly needed for such a small model. As stated above, both experiments differed in the TCA module. The difference in time shows the cost of including our module to deal with large models. We can clearly see that the GPU handles the additional operations without a noticeable increment of time.

The last experiment ($512^2 \times 1024$) used a large model that cannot be encoded on one mosaic in a conventional way. Therefore, our TCA module is mandatory to render it. In this case, the four color channels of a texture were used, and thus, the model was four times larger than the one used in the previous experiments. This experiment showed that the rendering time is greater than in the previous experiments. This result is somewhat a surprise, since the texture dimensions, operations and texture fetches are the same. In fact, the only difference is the number of color channels.

5 CONCLUSIONS

Due to the today's mobile GPU limitations, it was not possible to render volumetric models larger than 128^3 voxels on devices such as the Apple's smartphones and tablets. However, in this paper we have proposed a novel technique that enables mobile devices to render very large volumes by using multi-texturing to encode volumetric models on a set of RGBA 2D textures.

We have also proposed a simple and easy to implement architecture that can be used to adapt any existing di-

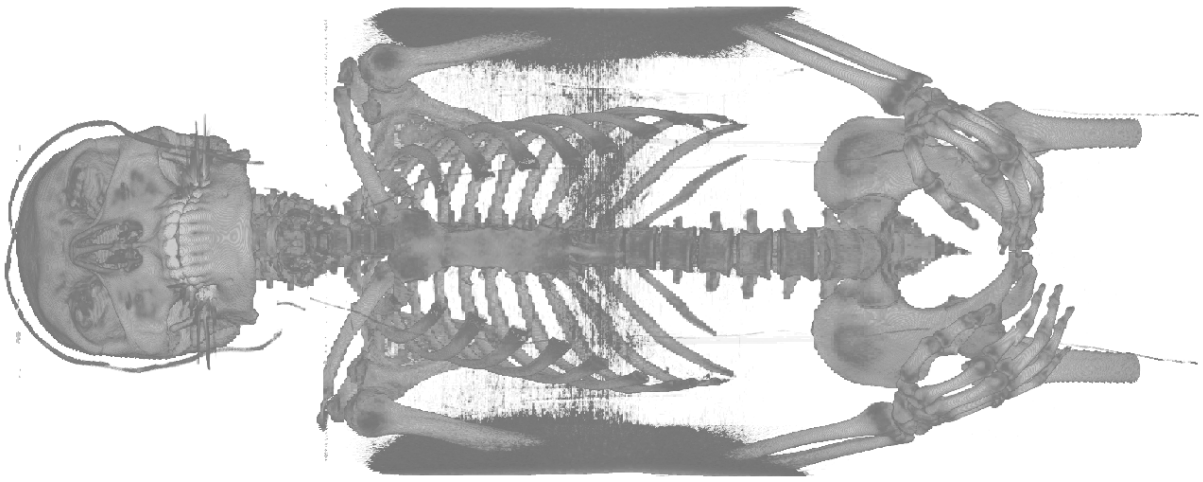


Figure 6: Visible human dataset rendered with WebGL, resolution: $512^2 \times 1024$ voxels.

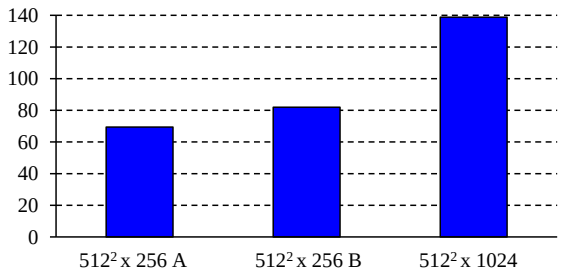


Figure 7: Rendering time (ms) for a desktop PC with a nVidia Geforce 8800GS. Screen resolution: 800^2 pixels. Raytracing: 128 steps.

rect volume rendering technique based on 3D textures to mobile devices and WebGL.

Our experiments have proved that we can render volumes of up to $512^3 \times 384$ voxels on a mobile device without decreasing the rendering speed. The proposed technique is also very akin to WebGL, because this standard shares the same limitations that mobile devices, mainly the lack of 3D texture support.

Regarding future works, we plan to study texture compression in order to reduce cache issues and improve efficiency. Furthermore, we plan to test the performance problems when transmitting large volumes across the Internet on WebGL. We want to explore multiresolution techniques in order to optimize network bandwidth. Progressive refinement techniques can probably be used in this context to improve the user interaction experience with the volume.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish "Ministerio de Ciencia e Innovación" and the European Union (via ERDF funds) through the research project TIN2011-25259; by the "Consejería de Innovación, Ciencia y Empresa" of the "Junta de Andalucía" and the European Union (via ERDF funds) through the research project P07-TIC-02773; and by the University of Jaén through the project PID441012.

6 REFERENCES

[1] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, and O. Ruiz. Interactive visualization of volumetric data with WebGL in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 137–146, New York, NY, USA, 2011. ACM.

[2] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf. Real-time volume graphics. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA, 2004. ACM.

[3] S. Guthe and W. Straßer. Real-time decompression and visualization of animated volume data. *Proceedings of the IEEE Visualization Conference*, pages 349–356, 2001.

[4] M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski. Advanced illumination techniques for GPU-based volume raycasting. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, pages 2:1–2:166, New York, NY, USA, 2009. ACM.

[5] ImageVis3D. ImageVis3D: A real-time volume rendering tool for large data. scientific computing and imaging institute (sci), 2011. [accessed 29 September 2011].

[6] F. Ino, S. Yoshida, and K. Hagihara. RGBA packing for fast cone beam reconstruction on the GPU. In *, Proceedings of the SPIE Medical Imaging*, 2009.

[7] S. Jeong and A. E. Kaufman. Interactive wireless virtual colonoscopy. *The Visual Computer*, 23(8):545–557, 2007.

[8] J. Kruger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[9] F. Lamberti and A. Sanna. A solution for displaying medical data models on mobile devices. In *SEPADS'05*, pages 1–7, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[10] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8:29–37, May 1988.

[11] M. Moser and D. Weiskopf. Interactive Volume Rendering on Mobile Devices. In *Workshop on Vision, Modelling, and Visualization VMV '08*, pages 217–226, 2008.

[12] A. Munshi, D. Ginsburg, and D. Shreiner. *OpenGL(R) ES 2.0 Programming Guide*. Addison-Wesley Professional, 1 edition, 2008.

[13] J. Noguera, J. Jiménez, C. Ogáyar, and R. Segura. Volume rendering strategies on mobile devices. In *International Conference on Computer Graphics Theory and Applications (GRAPP 2012). Rome (Italy)*, pages 447–452, 2012.

[14] Power VR. PowerVR Series5 Graphics SGX architecture guide for developers, 2011.

[15] J. Schneider and R. Westermann. Compression domain volume rendering. *Proceedings of the IEEE Visualization Conference*, pages 293–300, 2003.

[16] S. Thelen, J. Meyer, A. Ebert, and H. Hagen. Giga-scale multiresolution volume rendering on distributed display clusters. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6431 LNCS:142–162, 2011.

[17] B. Tomandl, P. Hastreiter, C. Rezk-Salama, K. Engel, T. Ertl, W. Huk, R. Naraghi, O. Ganslandt, C. Nimsky, and K. Eberhardt. Local and remote visualization techniques for interactive direct volume rendering in neuroradiology. *Radiographics*, 21(6):1561–1572, 2001.

[18] L. C. Vera. Volumetric medical images visualization on mobile devices. Master's thesis, Polytechnic University of Catalonia, 2010.

[19] D. Weiskopf. *GPU-based interactive visualization techniques*. Mathematics and visualization. Springer, 2007.

[20] C. Wooley. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*, chapter 35, pages 557–571. Addison-Wesley Professional, 2005.

[21] K. Xie, J. Yang, and Y. , Zhu. Real-time visualization of large volume datasets on standard pc hardware. *Computer Methods and Programs in Biomedicine*, 90(2):117–123, 2008.

[22] H. Zhou, H. Qu, Y. Wu, and M. yuen Chan. Volume visualization on mobile devices. In *14th Pacific Conference on Computer Graphics and Applications*, pages 76–84, 2006.

Voxel-Space Shape Grammars

Zacharia Crumley
University of Cape Town
South Africa
zacharia.crumley@gmail.com

Patrick Marais
University of Cape Town
South Africa
patrick@cs.uct.ac.za

James Gain
University of Cape Town
South Africa
jgain@cs.uct.ac.za

ABSTRACT

We present a novel extension to shape grammars, in which the generated shapes are voxelized. This allows easy Boolean geometry operations on the shapes, and detailing of generated models at a sub-shape level, both of which are extremely difficult to do in conventional shape grammar implementations. We outline a four step algorithm for using these extensions, discuss a number of optional enhancements and optimizations, and test our extension's performance and range of output. The results show that our unoptimized algorithm is slower than conventional shape grammar implementations, with a running time that is $O(N^3)$ for a N^3 voxel grid, but is able to produce a broad range of detailed outputs.

Keywords:

procedural generation, shape grammars, voxels

1 INTRODUCTION

For video games, virtual environments, and cinema special effects, cost-effective content creation is an increasing concern. The amount of models, animations, textures, and sounds needed for these applications has been steadily growing with the increase in computational power and the quality of graphics. It is now at a point where hundreds of modellers, animators, and artists will work for months or years to create the content necessary for a single mainstream video game or blockbuster film. The large size of these teams means the costs involved are significant, and in spite of the number of people working on the project, long development times are still the norm. For this reason, content creators have begun turning to *procedural generation*, as a way of decreasing costs and shortening development times.

Procedural generation refers to methods designed to algorithmically generate content, instead of having it hand-crafted. Minimal human interaction is required – generally limited to setting the initial parameters of the algorithm, or providing example inputs.

Today procedural generation is increasingly used to generate large amounts of high quality content, particularly plants, landscapes, and textures. This is evidenced by the growth of commercial procedural generation software, such as SpeedTree[1], Terragen[2], and CityEngine[3]. There are many procedural generation algorithms [14] but our research focuses specifically on *shape grammars* [18]. These are a type of formal grammar, consisting of an axiom (the initial item to begin with) and a set of production rules which modify, add, or replace items.

Shape grammars are distinguished from conventional grammars, in that their rules operate directly on geometric shapes instead of symbols from an alphabet. Their production rules include geometric operations on shapes (such as rotation, scaling, etc.), in addition to shape replacement (as grammars do with symbols). An example of a basic shape grammar is shown in figure 1.

Shape grammars were originally developed in architecture as a tool for formalizing architectural design. Although still used for that purpose, they are also finding use in computer science as a method for procedurally generating models of buildings and other structures. This is the application our research focuses on.

Conventional shape grammars operate on mesh representations of shapes. These are moved, rotated, subdivided and otherwise operated on until a final collection of shape meshes is produced: the output of the shape grammar.

However, there are two major problems with conventional mesh-based shape grammars, as used in procedural generation and both are difficult to solve.

Firstly, it is difficult to robustly apply constructive solid geometry (CSG) or Boolean geometric operations on

[1] http://www.speedtree.com/
[2] http://www.planetside.co.uk/
[3] http://www.procedural.com/

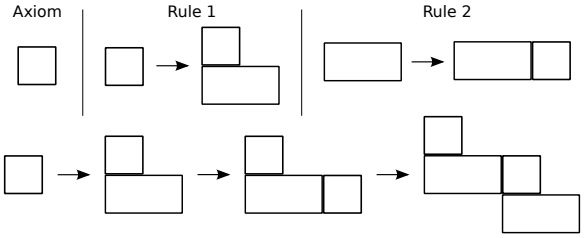


Figure 1: A simple shape grammar that produces an infinite series of 2D stairs, and its first three iterations.

meshes: overlapping edges and vertices must be identified and trimmed or removed, whichever is appropriate. This can lead to complications around numeric stability, slivers, degenerate triangles, and other issues. Alternatively, this redundant geometry can be left hidden, but this is inefficient. It may also be necessary to create new geometry. This process is complex, and there are edge cases that remain problematic. Even with such an algorithm, the overlap between different shapes can lead to texture seams. This is visually unappealing and difficult to overcome.

The second limiting problem of mesh shape grammars is that texturing is done at a per-shape level. The faces of each shape, or pre-made piece of geometry, have fixed texture coordinates and associated 2D textures that are projected onto the faces. This means it is difficult to have texture details that span multiple shapes, or control the textures at a sub-shape level.

For example, it would be difficult to create racing strips running along a car model produced by a shape grammar, since the different sections of the car are made up of different polygons created by different rules.

Modifying shape grammars to operate in a voxel-space solves these limitations. CSG operations on voxels are trivial, solving the first deficiency. For the second, textures can be assigned on a per-voxel basis, which allows details to more easily span shapes and removes the texture's association with a specific shape.

Using voxels presents some challenges, such as the need for large amounts of memory and storage, and the discrete nature of the underlying grid, which can introduce aliasing artifacts and other sampling issues. However, these problems can be dealt with, or worked around, using tree data structures for efficiently managing space, which also allow us to use high resolution voxel grids, reducing the impact of aliasing artifacts.

This paper presents our preliminary research into interpreting shape grammars in a voxel space, in contrast to the traditional mesh geometry approach. Our goal is to extend the expressive range of shape grammars with the ability to easily and robustly apply Boolean geometry operations.

Our major, and novel, contribution is an algorithm for this process. The algorithm is made up of four main stages, and produces a mesh model, suitable for use in real-time, or offline, 3D graphics. We also discuss optimizations and optional steps for the algorithm, as well as testing its performance and range of output.

2 RELATED WORK

Stiny and Gips [18] first developed shape grammars in an attempt to formalize architectural design. Although still used in architecture, they have also been adapted for use in the procedural generation of structures for other applications [19, 12].

Early shape grammars were simplistic with a limited range of output, but over time several ideas from procedural generation, most notably from L-systems, were incorporated into shape grammar implementations. Most notably, environmental sensitivity and stochastic rules [3]. With environmental sensitivity, rules can query the current set of shapes and adjust their output based on the information they get. Stochastic rules introduce randomness, by randomly choosing different outputs and parameters to introduce variation in the shapes generated.

Shape grammars were later extended to building generation using *split grammars* [19], which focus on subdivision of shapes. For example, a building's wall is divided first into floors, then the floors into different rooms, and finally the walls of the rooms into different windows. Split grammars are particularly well suited to façade generation [19] due to the regular, grid-like layout of building windows. They work by recursive subdivision of a shape, guided by following productions from a rule set. The final set of shapes that arise from the repeated subdivision form a model of the desired building wall. However, split grammars are less successful at creating internal structure. Early approaches to this problem tended to be simplistic, such as using n-sided prisms as the split grammar axioms [4]. Subsequently, better methods were developed, such as starting from the building's footprint obtained from aerial imagery [7], and creating the building's structure with shape grammar rules [12].

Along with the other extensions mentioned, the features of split grammars have since been incorporated into current grammar implementations, unifying all features under one grammar, for greater ease-of-use [12].

The range of output possible from modern shape grammar implementations is extremely broad [12] and as a result, shape grammars are considered the industry standard for procedurally generating architecture. They are able to create whole cities with realistic buildings (using additional techniques [15] to create the road networks and block layout). The best example of this in

practice is the CGA grammar of CityEngine[4], which procedurally creates cities and buildings.

Example-based shape grammars can be used to generate different models in the same style as the given-example. This can be done from an image of a building's façade [13], or from existing models [2].

The problem of easily, and visually, editing shape grammars has also been addressed by Lipp et al. [9] in their work on interactively editing shape grammars for architecture. Their approach is primarily concerned with operating on the grid-like façades of buildings, but does feature methods for assisting in the overall building structure creation.

Recent work [1, 5] has also developed methods for automatically creating a structural skeleton for models generated by shape grammars. These skeletons can then be used to create animations or run structural simulations on the generated models using a physics engine.

However, shape grammars do have limitations. In split grammars, shapes that span multiple subdivisions, and shape intersections, are difficult to handle gracefully. In addition, particularly unusual building designs with complex elements, such as tunnels and interior hollows, are very hard to generate.

Voxels have seen previous use in procedural generation, predominantly in games [5] and terrain representation. However, 3D texture synthesis methods have been extended to create 3D models. Merell's algorithm [11] works by assigning a cuboid section of geometry to each voxel type, and then keeping a record of how these cubes of geometry can be placed adjacent to each other while keeping the resulting model consistent. Texture synthesis methods are employed to create a, potentially infinite, voxel grid that corresponds to a consistent model. One downside is that this requires the manually created cuboid sections of geometry.

Another modelling approach that allows Boolean geometry operations while avoiding the issues around mesh-based CSG was proposed by Leblanc et al. [8]. Their approach allows modeling of shapes using Boolean geometry operations, but is substantially more complex than a voxel-based approach, and does not solve the issue of creating surface detail at a sub-shape, or trans-shape, level.

3 FRAMEWORK

The process of interpreting a shape grammar to produce a model in our framework consists of five stages (one of which is optional), and requires two inputs from the user. The final output is a textured mesh, suitable for use in modern graphics applications. The two user inputs are:

[4] http://www.procedural.com/

[5] http://www.minecraft.net/

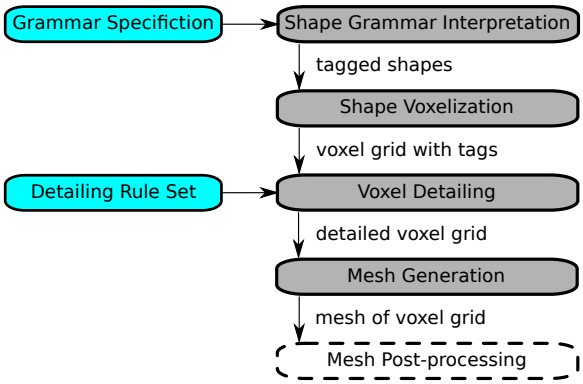


Figure 2: A flow chart of our algorithm. The cyan boxes are user inputs; the grey boxes are the four stages of the algorithm. Arrow labels show the output of each stage. Mesh post-processing is optional.

Shape Grammar Specification: This is the set of productions and associated parameters for the shape grammar itself.

Detailing Rule Set: The collection of rules that are used to assign visual detailing information to the voxel grid. This information is used when displaying the created model.

The five stages of the algorithm are:

1. **Shape Grammar Interpretation:** The input grammar is run to produce a collection of shapes.
2. **Shape Voxelization:** The shapes from the previous step are voxelized into a voxel grid.
3. **Voxel Detailing:** Surface voxels in the grid are assigned visual detailing information.
4. **Mesh Generation:** A mesh representation of the voxel shape is produced, using the marching cubes algorithm.
5. **Mesh Post-processing** (optional) The generated mesh is smoothed and refined.

Below, we cover each of the five stages in detail, explain their operation, what inputs they use, and what outputs they generate. Figure 2 shows an overview of the process.

3.1 Shape Grammar Interpretation

The first stage of the algorithm requires that we specify a shape grammar and then iterate it to produce the output set of shapes. This step is very similar to applying a conventional shape grammar, with some minor differences. A set of shape grammar rules, and an axiom shape, are provided by the user. This rule set is then run on the axiom, producing new shapes and modifying existing shapes on each iteration of the rules. This continually-updated set of shapes (the current shape set) converges to the final output of the shape grammar. An

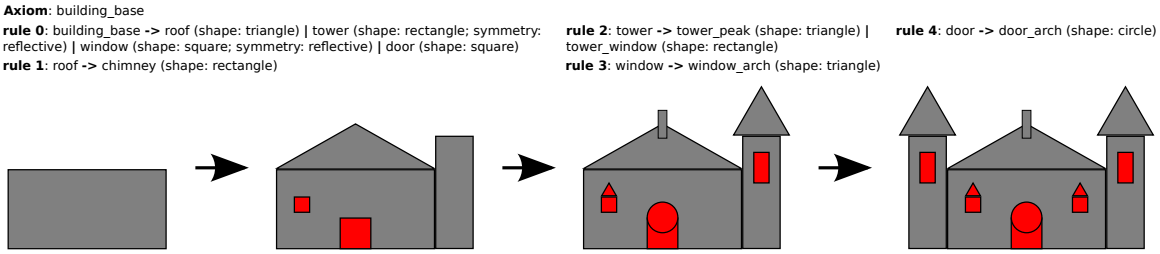


Figure 3: A simple shape grammar being interpreted in parallel to form a basic building. Grey shapes are additive, and red ones are subtractive. Position and size information in the example grammar are not shown, for the sake of simplicity. Note that the final step involves the symmetric copies being created, as the grammar only requires two iterations to complete.

Algorithm 1 Shape Grammar Interpretation

```
currShapeSet ← {Axiom}
iterations ← 0
while iterations < MaxIterations and currShapeSet.hasNonTerminals() do
  if parallelExecution = TRUE then
    for all i ∈ currShapeSet do
      i ← doRuleDerivation(i)
    end for
  else
    a ← getFirstNonTerminal(currShapeSet)
    a ← doRuleDerivation(a)
  end if
  iterations ← iterations + 1
end while
for all i ∈ currShapeSet do
  if i.hasSymmetry() then
    s ← i.createSymmetricCopies()
    currShapeSet.insert(s, i − 1)
  end if
end for
```

example of this stage is shown in figure 3. Pseudocode for this process is shown in algorithm 1.

This process will terminate under one of two conditions: Either after a user-defined maximum number of iterations, or once all shapes are terminal and none of the rules can be performed on the set of shapes.

Rules can be interpreted in parallel (as done in L-systems [16]), or in series (as done in traditional formal grammars [12]). These are appropriate in different situations, depending on the type of model being generated by the shape grammar. Serial rule derivation is suitable for most situations, except for models that have fractal qualities, where parallel rule derivation is advised.

Any of the many enhancements and extensions to grammar generation methods can be used here: environmental sensitivity, stochastic rules, a derivation tree for querying earlier shape set states, and more.

Our implementation includes these three extensions, as well as split grammar operators [12]. For more information on these extensions, we refer readers to the literature [16, 12, 3]. We also have a collection of standard utility functions for common operations, such as scaling, translation, rotating, and hollowing shapes. All that is required from this stage of the generation process is a specification of the final set of shapes.

There are two grammar extensions that we found very useful for generating models in our experiments.

First is the tagging of shapes with metadata. One of the operations that the shape grammar rules can perform is to add metadata tags to shapes in the current shape set. We implemented these as arbitrary strings. These serve to preserve additional information about the shapes that can be used in later stages of the generation algorithm. For example, in generating a castle, a cylinder (and its children if recursive tagging is used) could be tagged with “type:tower” and “material:stone”. These tags indicate additional properties of the shape that enhance later detailing and texturing.

Secondly, we support the specification of symmetry in the grammar rules. Our grammar implementation has special operators for indicating that a shape, or group of shapes, (and any child shapes that derive from them) should be cloned to create symmetrical versions.

We support two types of symmetry: rotational and reflective. In rotational symmetry, three arguments are provided to the operator, from which positioning information for the symmetrical copies can be derived: P_{rot} - the center point around which the symmetric branches are rotated; V_{rot} - a vector normal to the plane of rotation, and N_{rot} - the number of rotational copies to create.

For reflective symmetry, only two arguments are required to fully construct the mirror copies of the shape, or group of shapes, to be reflectively copied: P_{ref} - a point on the plane of reflection, and V_{ref} - a normal to the plane of reflection.

Symmetry information is specified when the grammar is run, but symmetric copies are only added once the grammar rules terminate. This is done as a post-process because further shapes could be added to the set of shapes undergoing symmetry, in iterations after the

symmetry is specified. Rather than tracking the symmetric copies and updating each of them for every change in shape, we simply flag the set of shapes for symmetry and wait until the rule derivation completes, before creating the symmetric copies.

Once the shape grammar has finished, a full specification of the final output set of shapes is passed to the next stage. This includes positions, dimensions, orientations, tags, and any other relevant information.

3.2 Shape Voxelization

In this phase, the shapes output from the shape grammar are voxelized into a voxel grid. This is analogous to rasterizing vector graphics into a pixel format. An example of this process is shown in figure 4, and pseudocode in algorithm 2.

Algorithm 2 Shape Voxelization

```
shapes ← getShapeGrammarOutput()
shapes ← sortByPriority(shapes)
shapes_bbox ← getBoundingBox(shapes)
gridResolution ← getVoxelGridResolution()
voxelGrid ← initializeEmptyGrid(gridResolution)
for all i ∈ shapes do
    i ← scaleShape(i, gridResolution, shapes_bbox)
end for
for all i ∈ shapes do
    voxelGrid.voxelizeShape(i)
end for
return voxelGrid
```

Due to the large memory requirements of storing voxel grids naïvely, it is infeasible to store the grid as a 3D array. Our implementation uses an *octree*, to efficiently manage space [17].

It is possible to use other tree data structures for storing the voxel grid, such as point region octrees (PR-octrees), kd-trees, or R-trees. In the general case, where no assumptions can be made about the data sets to be stored, and no special look-ups are required, the best option is an octree [17]. This is because the other tree types all require re-balancing (an expensive operation).

Tags associated with the shapes to be voxelized are assigned to the relevant voxels. In the case of overlapping shapes, it is possible that a voxel may inherit tags from multiple shapes. This is not problematic at this stage, but may cause ambiguities during detailing, which could have unintended consequences. Users should bear this in mind when designing grammars.

The order in which shapes are added is also important, because shapes may be additive or subtractive (additive for creating solid structures, or subtractive for carving empty spaces out of solids). Adding and subtracting geometry in this manner is not commutative. Hence a grammar may generate unintended results, depending on the order in which the shapes are voxelized.

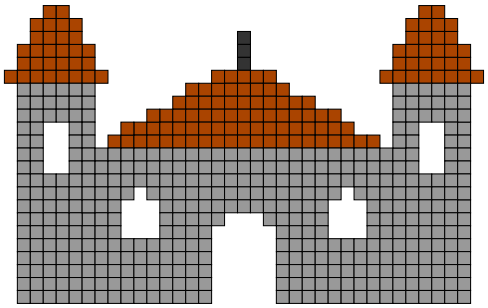


Figure 4: The output of the simple building shape grammar from figure 3 after being voxelized. The colours of the voxels correspond to the tags they inherited from the shapes. Grey indicates 'material:wall', brown indicates 'material:roof', and the dark grey 'material:chimney'.

To resolve this ambiguity, the shape grammar can assign a *priority* to the shapes. This is an integer that determines when the shape will be voxelized. Before voxelization, the shapes are sorted by priority, and added in sorted order. This allows a user to control when shapes are added, and resolve order-dependency issues.

Finally, we can manually edit the voxel grid once the shapes have been voxelized. This could be done to allow hand-crafted modifications to the output of a grammar, or because the user is dissatisfied with some aspect of the output that is difficult to correct in the grammar.

Manual editing is important for artists and modellers, and our shape grammar extensions do not restrict it at all, although it requires voxel editing software.

The final output of this stage is a 3D voxel grid, where each voxel is either solid or empty, and may have metadata tags associated with it.

3.3 Voxel Detailing

In this stage of the algorithm, voxels are assigned an appearance in the final model. This can include, but is not limited to, texturing information, bumps maps, displacement maps, lighting information, and materials. This is done on a per-voxel basis, by a user-created *rule set* which operates on each voxel individually. These rules may iterate over the voxels multiple times, allowing the creation of complex multi-pass detail. For example, cellular automata patterns could be created, since they map very well onto the discrete, gridded nature of voxels. The scope of these rules is extremely broad, and features such as context-sensitivity and randomness can easily be included. Everything from assigning a simple texture based on position, to randomized complex multi-pass procedural methods are possible. A simple example of a voxel grid after undergoing detailing is shown in figure 5, and pseudocode of the detailing process is found in algorithm 3.

Detailing Rules
rule_0: **if** "material:roof" in tags **then** texture = red_tile
rule_1: **if** "material:chimney" in tags **then** texture = black_stone
rule_2: **if** "material:stone" in tags **then** texture = random_selection(grey_brick, dark_brick)
rule_3: **if** voxel.borders_enclosed_space() == true **then** texture = blue_paint

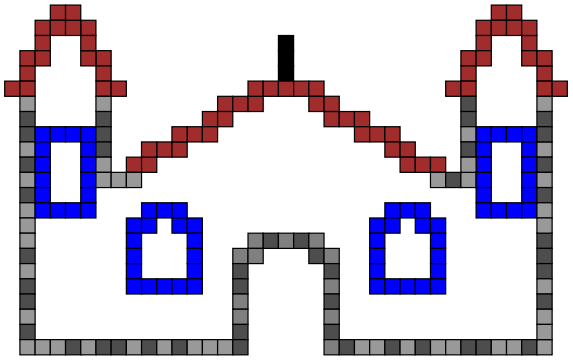


Figure 5: The voxel grid from figure 4 after detailing. Each voxel has been assigned a texture in accordance with the detailing rule set supplied. Non-surface voxels are ignored, and are not displayed in the diagram.

Algorithm 3 Voxel Detailing

```
detailingRuleSet ← getDetailingRuleSet()
maxIterations ← getMaxDetailingIteration()
surfaceVoxels ← voxelGrid.getSurfaceVoxels()
for i = 1 to maxIterations do
  for all j ∈ surfaceVoxels do
    n = voxelGrid.getNeighbouringVoxels(j)
    j.detailTags ← detailingRules.runRules(j, n)
  end for
end for
```

Detailing is done on a per-voxel level as opposed to the per-shape level of conventional shape grammars because this allows more complex procedural detailing of generated models, and it is much easier for detail features to span shapes and work on sub-shape scales. This also circumvents the problem of texture seams between adjacent shapes prevalent in conventional grammars. The disadvantage to this freedom is more complexity for the user. This complexity could be reduced in two ways. Firstly, by creating a visual rule editor to use, as opposed to text-based programming. Secondly, by designing an interface that allowed rapid prototyping of rules on small examples, to quickly detect problems. However, we did not implement these, and leave them to future work.

Relevant details about the voxel are passed to the rule set. In our implementation these details are: tags associated with the voxel; normal of the voxel; the maximum resolution of the octree; the coordinates of the current voxel; the count of the current iteration of the rule set, and the above details for all neighbouring voxels, within a user-specified radius.

It should also be noted that this stage is independent of previous steps. A detailing rule set can be applied to any voxel grid, and does not need to concern itself with how that data set was produced.

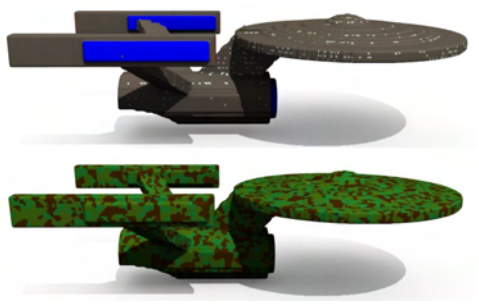


Figure 6: Our Enterprise model with two different detailing rule sets applied to it. Above, with its original detailing; below, with a camouflage pattern. This shows how detailing rule sets that are not reliant on shape tags, such as camouflage, can be applied to any voxel grid.

It is possible to have a detailing rule set that is completely independent of metadata tags. For example, detailing that creates a consistent pattern across the entire model without using the context information from the tags. These detailing rules will work on any model provided to them, regardless of tags. An example of such a detailing rule set being applied to a model intended for a different rule set is shown in figure 6.

However, in most practical situations, we expect that rules from the detailing rule set will be dependent on metadata tags in the voxel data set. For example, detailing a house's walls with brick textures and the doors with wooden ones requires that the two parts of the model be distinguished. Hence the user should ensure that the shapes in the shape grammar are properly tagged for the detailing stage.

Running snippets of code for each voxel in a grid can be extremely slow, especially so if the grid is large, or the rule's code is complex. For this reason, we only run the rule set on surface voxels in the grid.

We define a surface voxel to be any solid voxel in the grid which is 26-connected to at least one empty voxel. Because the marching cubes algorithm does not generate triangles for completely empty or solid space, only voxels on the border between solid and empty space will affect the final model. All others can be ignored.

Using the hierarchical structure of the octree, surface voxels can be quickly identified. If a node of the octree does not have any children, then only the voxels around the edge of that node need be checked further. For all reasonable models, this dramatically reduces the number of surface voxel candidates that need to be checked, improving speed by an order of magnitude or more.

The final output of this step is a voxel grid where all surface voxels have been assigned detailing information. This information must unambiguously provide all

information required for rendering, either as is, or when converted to a mesh.

3.4 Mesh Generation and Post-processing

There are many methods for rendering voxel grids. These are often based on ray tracing, or point rendering. In some situations it may be suitable to render the output of the shape grammar with these methods, but most graphics applications today work with triangle meshes, not voxel data sets. For this reason, we need to convert our voxel data set into a mesh that can be used in conventional raster graphics applications, such as modern 3D game engines.

The marching cubes algorithm [10] is a well-established solution to the problem of extracting a mesh representation of a voxel grid or isosurface. We make use of it to produce a mesh version of our generated model.

The algorithm outputs a list of triangles, each of which can be associated with a voxel in the input grid. Each triangle is then assigned textures, materials, and other detailing information from the surface voxel.

Hence we end up with a fully textured and detailed mesh representation of the voxel grid that the original shape grammar produced.

The mesh produced by the marching cubes algorithm is suitable for direct use in graphics applications, but its visual quality could be improved by post-processing.

One of the problems with the marching cubes algorithm is that the output mesh has visual artifacts caused by the discrete nature of the voxels. Curves in particular, are not fully captured during the meshing process, and will instead appear 'bumpy', although increasing the resolution of the voxel grid can reduce this.

The severity of this problem can be reduced by an appropriate mesh smoothing algorithm, which will significantly decrease the impact of such artifacts [6].

It should be noted though, that naïve smoothing algorithms can lose details that are not artifacts, and should be retained, such as sharp corners. For this reason, we recommend the use of one of the more advanced smoothing algorithms, which will retain these features. There are a number of such algorithms, but in general these advanced methods of smoothing come at the cost of more complexity and a longer running time.

3.5 Optimizations

There are several optimizations to our algorithm that we did not implement due to time constraints. These have the potential to dramatically reduce running times, and hence we discuss them here.

Voxelization of shapes can be performed extremely efficiently by exploiting the hierarchical nature of the octree. Beginning at the root of the tree, query the intersection between each of the eight children of the octree node, and the shape to be added. If the area covered by a child node is entirely within the shape, then that voxel is set with the shape's information, if there is a partial intersection between the child node and the shape, then the algorithm is recursively called on that node.

While faster, this is more complex to implement, and requires an exact collision detection algorithm. We suggest that future implementations make use of this method to greatly reduce run times.

The implementation of marching cubes can also be substantially accelerated by only marching over the surface voxels of the voxel grid. Since solid or empty regions will not produce triangles, the voxels of those regions need not be processed. The list of surface voxels from the detailing step can be re-used here.

4 TESTING AND EXPERIMENTATION

In order to evaluate our voxel-space extensions to shape grammars we undertook three experimental tasks: performance testing, where we analyzed the time and memory required; variation testing, where we produce multiple similar models from a single grammar; and output range testing, where we examine the range of outputs our algorithm can produce, and its ability to generate models of well-known structures.

4.1 Performance

We analyzed our algorithm's performance across a variety of voxel grid sizes and user inputs. The two main results of interest are the time taken to generate a model, and the peak memory usage of the process.

We decomposed timing into the four stages of the algorithm to get an idea of their relative durations (post-processing was excluded as it is an optional step).

Testing was performed on a PC with an Intel Core 2 Duo clocked at 2.4Ghz and 3 gigabytes of RAM.

Performance testing was conducted with a selection of 36 shape grammar and detailing rule-set combinations, at 4 different voxel grid resolutions. The selection of grammars and rule sets was specifically chosen to encompass a wide range of complexity. figure 7 shows the timing results across all of the 36 models.

Before analyzing the results it should be noted that our implementation was strictly intended as a single-threaded proof-of-concept. Hence, performance was not a priority, and there is large scope for improvements in this area (as mentioned in section 3.5.) Nonetheless, we include our results as we believe they provide a baseline for comparison to future implementations of our work.

The first thing to note is that the shape grammar interpretation is orders of magnitude faster than the other

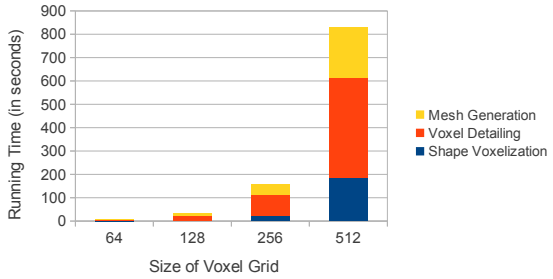


Figure 7: A cumulative graph of the average times taken for our algorithm to run on 36 different inputs, covering a range of complexities. Shape grammar interpretation is not shown as it was negligible compared to the other three stages.

stages, due to its independence from the voxel grid resolution. The average interpretation time was 50 milliseconds. Due to the minuscule relative time, grammar interpretation is not shown in figure 7.

As expected the running times of the other stages of the algorithm is approximately cubic in the size of the voxel grid. This is expected, as their running time is directly proportional to the number of voxels to operate on, which scales cubically with the size of the grid.

The biggest cause of variation in running times is the number of iterations in the surface detailing. Because each voxel must be processed for every iteration, the number of iterations makes a large difference in the amount of processing to be done, especially for higher voxel grid resolutions.

Peak memory usage followed the same pattern of being cubic in the resolution of the voxel grid. The minimum and maximum amount of memory used, across all testing inputs, were approximately 150 and 1400 megabytes, respectively.

These running times are significant for larger resolution grids. However, in practice, users can prototype their grammars and rule sets on lower resolution models and, once satisfied with them, then do off-line generation of a high resolution model for actual use. This means the long running times for large models will not significantly disrupt work-flow.

4.2 Variation and Range Testing

Variation testing involved randomizing the parameters in several of our shape grammars, and producing multiple models from them. The objective is to ensure that our algorithm is capable of producing many different models that share a similar style, from a single shape grammar. A selection of the models produced in this manner are shown in figure 8.

As can be seen from the images, our algorithm is capable of producing a variety of models, sharing a common theme and style, from a single shape grammar and detailing rule set, by randomizing the parameters of the shape grammar and detailing rules.

To test the power of our shape grammar extensions, we created shape grammars and detailing rule sets representative of a broad variety of models, including imitations of well-known existing structures. A selection of these generated models are shown in figure 9.

5 LIMITATIONS

There are two limitations to our voxel-space shape grammar algorithm that could restrict its potential uses.

Firstly, in order to obtain a high quality model from a voxel data set, the set must be at a high resolution, so as to remove “blocky” visual artifacts caused by the discrete nature of a voxel grid. Mesh smoothing as a post-process helps, but it is not sufficient on its own.

However, the higher the resolution, the slower the voxel detailing process is. This is because each voxel in the model must be detailed, and the number of voxels is cubic in the dimensions of the voxel grid.

Secondly, texturing at a per-voxel level may be insufficient in certain cases, such as for curved surface details, where the discrete nature of the underlying voxel data can cause visual artifacts. For example, an elaborate spiral design with fine curved detail on the side of a spaceship would almost certainly run into sampling issues if created with a detailing rule set.

A possible solution to this problem would be allowing the addition of decal textures to the final version of the mesh. These decals would replace the existing details in certain locations and display detail that could not be created within the detailing rule set framework.

It must be noted though, that neither of these limitations are critical, and none of them should be problematic in the majority of cases.

6 CONCLUSION

We have presented a novel extension to conventional shape grammars, where the shape output of the grammar is voxelized, allowing more robust Boolean geometry operations and a new per-voxel approach to detailing the surface of generated models.

These extensions address two shortcomings in current shape grammar implementations: the support of complex shapes through CSG, and sub-, or trans-, shape detailing at per-voxel level, for more elaborate and controlled texturing.

Our algorithm is slower and more memory intensive than conventional shape grammar implementations, but not outside acceptable limits. Additionally, our extensions allow the generation of a wide range of models, including variations from a single shape grammar.

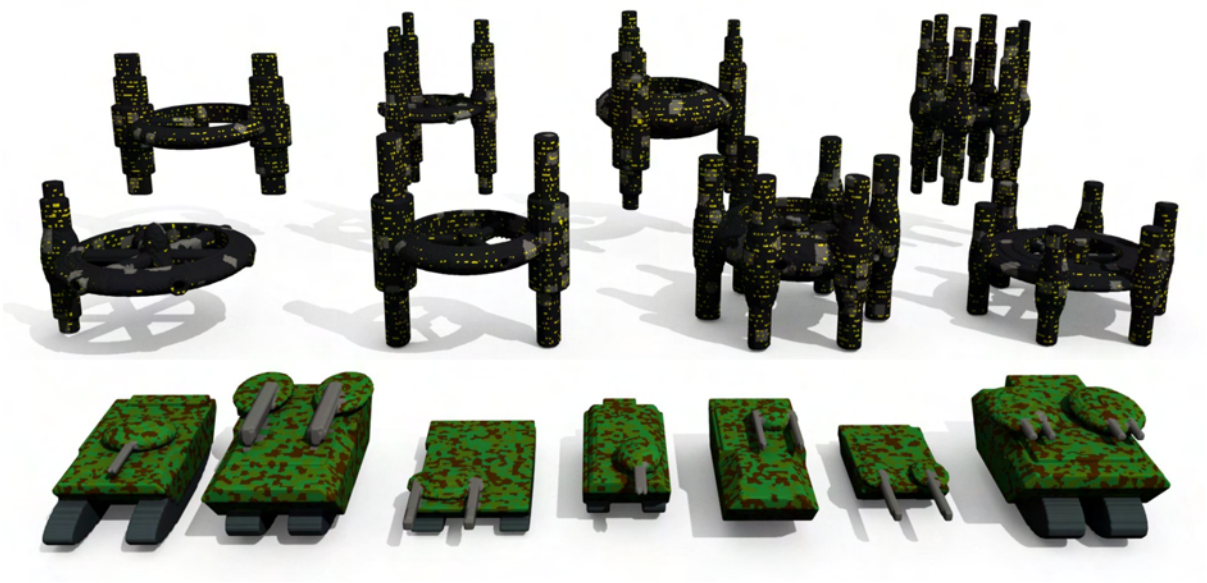


Figure 8: A selection of tanks and space stations produced by two of our shape grammars, using randomized parameters in their rules. This shows how a single grammar can produce multiple models in the same style. These models were all generated from cubic voxel grids of resolution 256.

Figure 9: A broad selection of the models produced by our algorithm, using a range of detailing rule sets and shape grammar extensions, including cellular automata patterns, symmetry and multiple-pass textures. All of these models were generated using a cubic voxel grid of size 256.

Our extensions add new functionality to shape grammars, without losing existing capabilities, and significantly increase the range of achievable content.

6.1 Future Work

The per-voxel detailing stage could be expanded to address the interior of generated models. In our work, we have only performed detailing on the surface voxels of the model, but the method could be extended to detail the interior voxels too. This would allow the creation of details such as rooms inside generated buildings.

Post-processing could also be done on the voxel grid before it is detailed. This could be used to add, remove, and tag voxels to create detail corresponding to damage, wear and tear over time, growth of mold, and more.

The voxel detailing rules could be extended to operate at multiple resolutions of the voxel grid. Octree nodes could easily be coalesced to form a lower-resolution version of the model, to which the rule set could then be applied. This would allow the creation of large scale detailing initially, working down to finer details as the rules are run at higher resolutions.

Finally, a solution to the constraint of texturing being limited to a per-voxel level is the use of decal textures on the generated mesh. Detailing could be extended to

allow arbitrary textures to be projected onto the generated mesh, complimenting the textures assigned in the detailing step. This would allow texturing beyond the per-voxel level our system is currently limited to.

ACKNOWLEDGEMENTS

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

Funding assistance for this research was also provided by the University of Cape Town.

7 REFERENCES

[1] Richard Baxter, Zacharia Crumley, Rudolph Neeser, and James Gain. Automatic addition of physics components to procedural content. In *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, AFRIGRAPH '10, pages 101–110, New York, NY, USA, 2010. ACM.

[2] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A connection between partial symmetry and inverse procedural modeling. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 104:1–104:10, New York, NY, USA, 2010. ACM.

[3] Peter Eichhorst and Walter J. Savitch. Growth functions of stochastic lindenmayer systems. *Information and Control*, 45(3):217–228, June 1980.

[4] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Real-time procedural generation of 'pseudo infinite' cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '03, pages 87–ff, New York, NY, USA, 2003. ACM.

[5] Martin Ilčík, Stefan Fiedler, Werner Purgathofer, and Michael Wimmer. Procedural skeletons: kinematic extensions to cga-shape grammars. In *Proceedings of the 26th Spring Conference on Computer Graphics*, SCCG '10, pages 157–164, New York, NY, USA, 2010. ACM.

[6] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 943–949, New York, NY, USA, 2003. ACM.

[7] R. G. Laycock and A. M. Day. Automatically generating large urban environments based on the footprint data of buildings. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, pages 346–351, New York, NY, USA, 2003. ACM.

[8] Luc Leblanc, Jocelyn Houle, and Pierre Poulin. Modeling with blocks. *The Visual Computer (Proc. Computer Graphics International 2011)*, 27(6-8):555–563, June 2011.

[9] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, New York, NY, USA, 2008. ACM.

[10] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[11] Paul Merrell. Example-based model synthesis. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 105–112, New York, NY, USA, 2007. ACM.

[12] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 614–623, New York, NY, USA, 2006. ACM.

[13] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 85, New York, NY, USA, 2007. ACM.

[14] F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling: a procedural approach*. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[15] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[16] P. Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[17] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[18] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. *Information processing*, 71:1460–1465, 1972.

[19] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 669–677, New York, NY, USA, 2003. ACM.

Interactively Simulating Fluid based on SPH and CUDA

Yige Tang
Beijing Normal University
No. 19 XinJieKouWai St
Haidian District
100875, Beijing, China
solidsnake1905@gmail.com

Zhongke Wu
Beijing Normal University
No. 19 XinJieKouWai St
Haidian District
100875, Beijing, China
zwu@bnu.edu.cn

Mingquan Zhou
Beijing Normal University
No. 19 XinJieKouWai St
Haidian District
100875, Beijing, China
mqzhou@bnu.edu.cn

ABSTRACT

In this paper, we propose a novel method of interactive fluid simulating based on SPH, and implement it on CUDA (Compute Unified Device Architecture). Firstly we use SPH (Smoothed Particle Hydrodynamics) theory to simulate the motion of fluids. Secondly we propose an interactive method between fluid and rigid objects. We treat the rigid objects as two different types, static one and dynamic one. We deal with the two types in separately suitable ways in order to enforce their motion similar to the real world. By taking advantages of CUDA which are greatly effective for large scale numeric computation in parallel, our simulation achieves real time with low cost.

Keywords

Simulation, Fluid, SPH, CUDA, Interactive

1. Introduction

As a common phenomenon in nature, simulation of fluid including water and smoke is an important part in visual reality. After Reeves' proposal of particle system in 1983, which is used to present non-solid objects such as water, fire and smokes, researchers have done a lot of work for simulating fluid vividly and effectively. Fluid simulation is usually applied in medical image visual reality and video game. Simulating fluid in computer is a tough issue. Although the theory of computational fluid dynamics has existed for many years, some properties of fluid which contains convection, turbulence and surface tension are difficult to be expressed through simple modeling. However, due to the fact that the real time simulation is more important than computational precision in computer graphics, the mathematical model and implementation focus on real time and visual effects more than precision in fluid computation.

The earliest approach of fluid simulation is simple particle system, which is proposed by Reeves in simulating fire and flake [Ree83a]. After that, Shinya and Stam improved particle system respectively in their work by introducing random turbulences [Shi92a] [Sta93a]. Fluid simulation based on Navier-Stokes Equations is implemented in 2D space firstly, both Gamito and Yaeger et al. have made contributions to it [Gam95a] [Yae86a]. In 1997, Stam et al. proposed an approach based on grid to simulating smoke [Sta99a], which is the first interactive method on fluid simulation.

There are two approaches of simulating fluid based on Navier-Stokes equations, the Eulerian viewpoint and the Lagrangian viewpoint.

In the Eulerian approach, which is based on

position, the fluid properties on some fixed points are computed. The absolute locations of fixed points never change, and the properties need to be computed are velocity, pressure, density, etc. The Eulerian method is appropriate for simulating gases, while is not able to present liquid well in wave and foam. The Lagrangian approach is different from Eulerian one as it is based on particles. Desbrun et al. and Tonnesen use particles to present soft objects [Ses96a] [Ton98a]. Witkin et al. use particles to control implicit surface [Wit94a]. Dan et al. simulate lava by using particles [Sto99a]. Comparing with Eulerian approach, Lagrangian approach has several following advantages. First, it's not grid based, so fluid can move in the whole scene and interact with other object. Second, it can present more details of fluid, such as the mergence and dispersing of water drop.

Recently, the most common Lagrangian approach is based on smoothed particle hydrodynamics (SPH), which is proposed by Lucy in 1977 [Luc77a] firstly used in astronomical phenomenon. Muller et al. introduced SPH theory to compute fluid simulation in 2003 [Mul03a]. Their approach simplifies solution method of Navier-Stokes equations, while the amount of calculation is so great that it's hard to implementing animation in real time when the quantity of particles is huge.

Development of programmable GPU technique makes large-scale numeric computation be solved effectively under low cost. Due to the feature, SPH method can be solved in parallel. Through programming on GPU, real-time simulation for large-scale-particles fluid becomes possible. The GPU based radix sorting approach designed by Satish et al. combining with spatial uniformed-grid, reduces the cost in finding neighbors of particles [Sat08a].Then the method improves the computational speed. In 2004 Amada et al. implemented forces' computation of particles [Ama04a], while the neighbor finding task is still done by CPU. The method completely implemented on GPU is firstly proposed by Kolb and Cunts in 2005 [Kol05a]. This approach emerges earlier than CUDA.

CUDA is a general-purpose GPU programming toolkit released by nVIDIA in 2007, which makes it possible to use C language program on GPU. With the help of CUDA, the processors of GPU can run parallelly by independently executing the same groups of operations on different sets of data. The above features are well suited for SPH method, because the same groups of operations such as force computation, speed and position update are completely same and have to be executed for each particle.

2. Fundamentals

2.1 Smoothed Particle Hydrodynamics

Essentially, SPH is a computational model to compute the interactive result of each particle in the fluid system. It defines a way to compute properties of a fix point impacted by other particles in the continuous space. A distance related weighted function $W(r)$ which is called kernel function is the key of SPH method.ris the distance between some position x and particle $i's$ position x_i. Another form of kernel function is $W(|x - x_i|)$. Kernel function satisfies the following equation $\int W(|x - x_i|)dx = 1$.We use poly6 kernel as our kernel function, its form is

$$W_{poly6}(x) = \frac{315}{64\pi h^9}\begin{cases}(d^2 - r^2)^3 & 0 \le r \le d \\ 0 & \text{otherwise}\end{cases} \quad (1)$$

After affirming kernel function, smoothed fields $A_s(x)$ of arbitrary attributes A_i of the particle as

$$A_s(x) = \sum_j m_j \frac{A_j}{\rho_j} W(|x - x_j|) \quad (2)$$

After replacing the kernel in formula (2) by the gradient of the kernel, we easily get the gradients of the field as following.

$$\nabla A_s(x) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(|x - x_j|) \quad (3)$$

In SPH simulation, we only need consider pressure, viscous force and external force. Pressure and viscous force can be calculated by above formulations.

2.1.1 *Pressure*

Formula (2) is used on computing pressure yields

$$f_i^{pressure} = -\nabla p(x_i) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(|x_i - x_j|) \quad (4)$$

If there are only two particles, the pressure force calculated by formula (4) will not be symmetric because the pressures at the locations of the two particles are not equal. Following equation is a simple, stable and fast solution.

$$f_i^{pressure} = -\nabla p(x_i) = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(|x_i - x_j|) \quad (5)$$

Since particles only carry the three quantities mass, position and velocity, the pressure at particle locations has to be evaluated firstly.
We compute pressure by using equation (6)

$$p = k(\rho - \rho_0) \quad (6)$$

Here k is a gas constant and ρ_0 is the environmental pressure.

2.1.2 *Viscosity*

We use following equation to calculate viscosity.

$$f_i^{viscosity} = \mu \nabla^2 v(x_i) = \mu \sum_j m_j \frac{v_j}{\rho_j} \nabla^2 W(|x_i - x_j|) \quad (7)$$

Since viscosity forces are dependent on velocity differences and not on absolute velocities, there is a natural way to symmetrize the viscosity forces by using velocity differences:

$$f_i^{viscosity} = \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(|x_i - x_j|) \quad (8)$$

2.1.3 *External Forces*

External forces include gravity, collision forces and interaction forces with environment. They are applied directly to the particles without the use of SPH.

2.1.4 *Computing Procedure*

The SPH model is executed under the following steps:

1) Find neighbors of each particle
2) Calculate the particle density
3) Calculate forces on particles
4) Update position and velocity of particles in next time step

After finishing these steps, the particles move obeying SPH rules can be rendered shown in Figure 1.

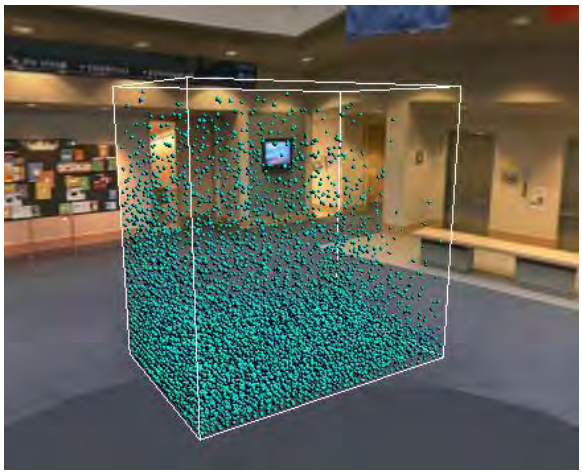


Figure 1. A group of SPH particles. Their motions are computed by above formulas.

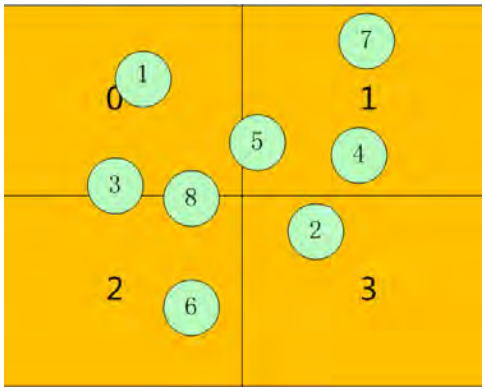
2.2 Neighbors Search

SPH is computationally heavy. The first step is finding neighbors of each particle. In worst case, each particle should be compared with all of others, whose complexity is $O(N^2)$.To avoid this, using uniform grid reduces most of cost. We use the algorithm presented in [Gre08a], which can be summarized as follows:

1) Divide the simulation domain into a uniform grid.
2) Use the spatial position of each particle to find the cell it belongs to.
3) Use the particle cell position as input to a hash function
4) Sort the particle according to their spatial hash.
5) Reorder the particles in a linear buffer according to their hash value.

After those steps, it satisfies that particles in the same cell will lie consecutively in the linear buffer, which makes finding neighbors much more effectively.

Radix sort's implementation on GPU is proposed by Satish et al. [5]



(a)

Before sort:

ParticleIndex

1	2	3	4	5	6	7	8

ParticleCell

0	3	0	1	1	2	1	2

(b)

After sort:

ParticleIndex

1	2	3	4	5	6	7	8

OldIndex

1	3	4	5	7	6	8	2

ParticleCell

0	0	1	1	1	2	2	3

CellStart

1	3	6	8

CellEnd

2	5	7	8

(c)

Figure 2. Data structure used by radix sort. (a) presents particles' position in the cells. (b)is status of arrays before sort. (c) shows the arrays needed after sort. The particles in same cell are consecutive in ParticleIndex array. We use OldIndex array to get the particle index before sort in order to access the velocity and position of particles.

3. Interaction with Rigid Objects

We classify rigid objects into two types. One type of them is static object such as glass with water. However the water acts, the glass keeps still. Another type is dynamic object such as a block on the water. We use different methods to simulate them respectively.

3.1 Interaction with Static Objects

It is easy to compute the interaction between particles and static rigid objects. Since the object can't move, there is no need to compute a force from the particles on the object. We only compute the penalty force, which forces the particle back into the fluid region in the opposite direction. Deformation will not happen on that static object.

When a particle collides with a static object, a penalty force is applied. Moore and Wilhelms provide a comprehensive introduction to the penalty force method [Mat88a]. Here we use the penalty force applied to a fluid particle can be calculated by equation (9).

$$f^{col} = k_s dn + k_d(v \cdot n)n \qquad (9)$$

In equation (9), d is the distance by which the particle has interpenetrated the static object, k_s is a

spring constant, k_d is a damping constant, n is the normal vector at the collision location, and v is the relative velocity of the particle to the static object.

3.2 Interaction with Dynamic Rigid Bodies

Computing interaction with dynamic objects is more complicated.

We treat dynamic rigid bodies as a portion of fluid whose initial density is greater than the initial density of the fluid. The only difference is in computing the pressure impaction between fluid and particles of rigid bodies. The rigid particles push the fluid away from the rigid bodies. Likewise, the fluid particles apply a pressure that results in a pure translation or rotation of the rigid bodies.

The pressure applied on a rigid body particle is given by equation (10), and the pressure at a fluid particle is given by equation (11).

$$p = \begin{cases} k^{rigid}\left(\rho - \rho_0^{rigid}\right) & \rho \geq \rho_0^{rigid} \\ 0 \text{ otherwise} \end{cases} \quad (10)$$

$$p = \begin{cases} k^{fluid}\left(\rho - \rho_0^{fluid}\right) & \rho \geq \rho_0^{fluid} \\ 0 \text{ otherwise} \end{cases} \quad (11)$$

Here, ρ_0^{rigid} is the rigid body's initial density, while ρ_0^{fluid} is the fluid's.

The rigid objects construct of rigid particles, whose motion is restricted to translation and rotation without deformation. Pressure is applied individually to each rigid particle, causing them to move independently. After that, we need modify the position of those particles to keep the rigid body's shape.

The mass M of whole object formed by rigid particles is calculated by formula (12).

$$M = \sum_j m_j \quad (12)$$

Here, m_j is the single particle's mass. Assuming all of the particles are the same, then the velocity of the center of mass can be computed by formula (13).

$$v_g = \frac{1}{N}\sum_j v_j \quad (13)$$

N is the amount of particles, and v_j is velocity of a single particle j.The angular velocity of the rigid body ω is approximated by equation (14).

$$\omega = \frac{1}{I}\sum_j q_j \times v_j \quad (14)$$

Here q_j is the location of a single particle j relative to its corresponding rigid body center of mass.

By using those above equations, we can get a rigid object's velocity and angular velocity. Then we can simulate the motion of that rigid object.

4. CUDA Computation

In our implementation, we use three texture arrays to store positions, velocities and densities of particles in last computing procedure, and we write new values to global memory of GPU respectively. The following Table 1 outlines the steps of our SPH algorithm with CUDA.

```
SPHCompute() //for each frame
{
    Copy particles' properties from CPU to GPU
    Set physical parameters of environment
    Hash particles by their spatial position
    Sort particles using radix-sort
    /*--ComputeDensity--*/
    Launch CUDA kernel function for each thread
    Each thread calculate one particle
    Compute new densities using other particles in 27 neighbor grids by using SPH kernel function
    __syncthreads()
```

/*--Compute Force--*/
Launch CUDA kernel function for each thread
Each thread calculate one particle
Calculate forces using densities computed above, including pressure and viscosity
Handle external forces such as gravity
}

Table 1.The procedure of SPH computing in our implementation

5. Rendering

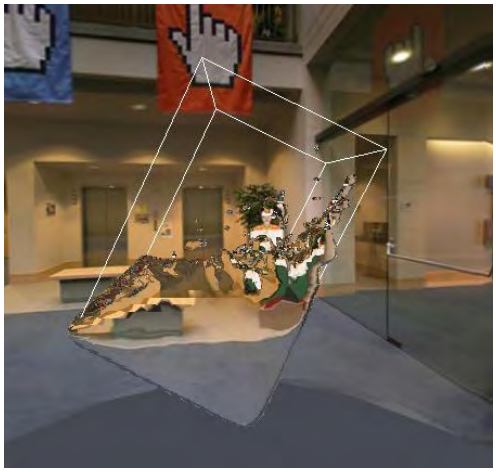
By using density we have computed in above work, the Marching Cubes method is applied in our work. An image spaced method is applied to simulate the refractive effect. The rendering method not only obtains a good visualization effect, but also bring little calculation burden. The rendering results are shown in Figure 3.

Figure 3. Water rendered by using marching cube. The refraction effects are generated by CGSL

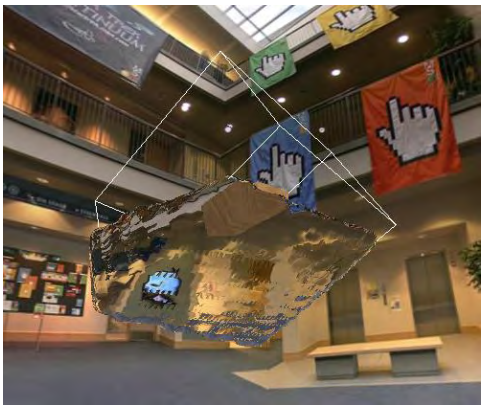
.

6. Conclusions

In this paper, we presented an interactive fluid simulating method based on Smoothed Particle Hydrodynamics. Our implementation can simulate fluid in real time and vividly with the help of CUDA and GLSL. The fluid we simulate looks like in real world with wave and foam. Additionally, the method which we propose in this paper also makes the fluid interacting with rigid objects well. The fluid in the box can move with the rotation of the box. The wood block in the box is floated under the force from the water. Figure 4 shows the interactive results of our implementation.



(a) Result of interacting with static rigid objects.



(b) Result of interacting with dynamic rigid objects.

Figure 4. Results of interacting with rigid objects

Through using CUDA, the method achieves real time simulation, since we take the advantage of the capacity of parallel computation afforded by GPU. Our implementation runs on below platform:

Windows7 OS 64-Bit， Intel(R) Core(TM) i7 CPU @3.07GHZ, 6GB RAM and GeForce GTX 570 with 1280MB video memory.

The results of frame rates are shown in table 2.

number of Particles	frame rate (FPS)
65,536	98.1
131,072	42.2
262,144	18.8

(a) results without free surface rendering

number of Particles	frame rate (FPS)
65,536	50.9
131,072	22.8
262,144	10.4

(b) results with free surface rendering

Table 2. Runtime result of our implementation on above platform

The large performance gap between the results in Table 2(a) and the ones in Table 2(b) is due to the surface rendering. The surface rendering takes extra cost on memory and time. The above table presents that our implementation performs very well.

7. References

[Ama04a] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, K. Chihara. Particle - based fluid simulation on the GPU. Proc. ACM Workshop on General - purpose Computing on Graphics Processors, 2004.

[Gam95a] M. N. Gamito, P. F. Lopes, and M. R. Gomes. Two dimensional Simulation of Gaseous Phenomena Using Vortex Particles. In Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation, pages 3–15. Springer - Verlag, 1995.

[Gre08a] S. Green. Cuda Particles. Technicle report, NVIDIA.

[Kol05a] A. Kolb, N. Cuntz. Dynamic particle coupling for GPU-based fluid simulation. Proc. 18th Symposium on Simulation Technique, 722-727, 2005.

[Luc77a] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. The Astronomical Journal, 82: 1013-1024, 1977.

[Mat88a] M. Matthew, J. Wilhelms, Collision Detection and Response for Computer Animation. Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press: p. 289-298, 1988.

[Mul03a] M. Muller, D. Charypar, M. Gross. Particle-Based Fluid Simulation for Interactive Applications.Eurographics/SIGGRAPH Symposium on Computer Animation, 2003

[Ree83a] W. T. Reeves. Particle systems: a technique for modeling a class of fuzzy objects. ACM Transactions on Graphics 2(2), pages 91-108, 1983.

[Sat08a] N. Satish, M. Harris, M. Garland. Designing efficient sorting algorithms for many core gpus. NVIDIA Technical Report NVR-2008-001, NVIDIA Corporation, Sept, 2008.

[Ses96a] M. Desbrun and M. P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In Computer Animation and Simulation ’96 (Proceedings of EG Workshop on Animation and Simulation), pages 61-76. Springer - Verlag, Aug 1996.

[Shi92a] M. Shinya and A. Fourier Stochastic Motion: Motion Under the Influence of Wind. In Proceedings of Eurographics’92, pages 119-128, September 1992.

[Sta93a] J. Stam and E. Fiume. Turbulent Wind Fields for Gaseous Phenomena. In Proceedings of SIGGRAPH ’93, pages 369–376. Addison-Wesley Publishing Company, August 1993.

[Sta99a] J. Stam. Stable fluids. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[Sto99a] D. Stora, P. Agliati, M. Cani, F. Neyret, J. Gascuel. Animating lava flows. In Graphics Interface, pages 203-210, 1999.

[Ton98a] D. Tonnesen. Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation. PhD thesis, University of Toronto, November 1998.

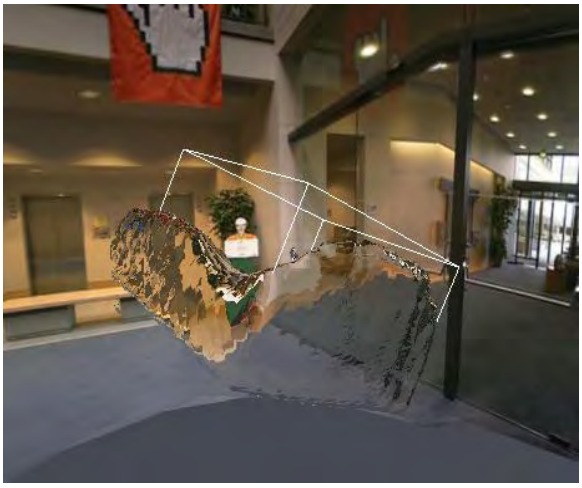
[Wit94a] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. In Computer Graphics (Proc. SIGGRAPH ’94), volume 28, 1994.

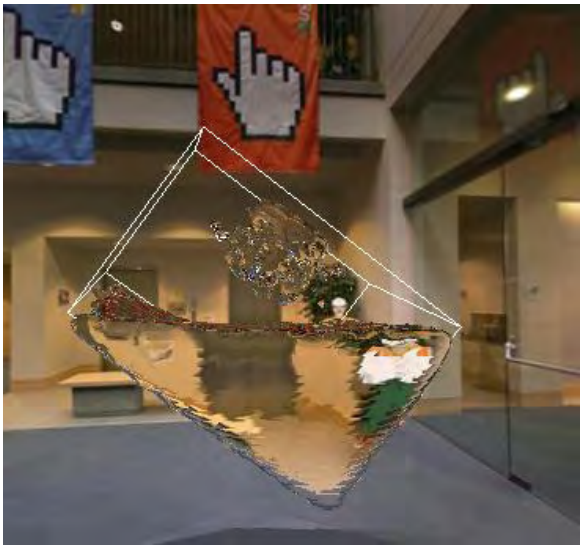
[Yae86a] L. Yaeger and C. Upson. Combining Physical and Visual Simulation. Creation of the

Planet Jupiter for the Film 2010. ACM Computer Graphics (SIGGRAPH ’86), 20(4):85–93, August 1986.

Acknowledgements

The work is partially supported by National Natural Science Foundation of China (No: 61170170) and the Fundamental Research Funds for the Central Universities (No: 2009SD-11)
Corresponding author, Email: zwu@bnu.edu.cn





Artificial jellyfish: evolutionary optimization of swimming

V. Lazunin
Hosei University
lazunin@gmail.com

V. Savchenko
Hosei University
vsavchen@hosei.ac.jp

ABSTRACT

Jellyfish, also known as "medusae", move by rhythmically contracting and expanding their bell-shaped bodies and are the earliest known animals to achieve locomotion through the muscle power. Development of a generalized dynamical model of medusan swimming is of interest to biologists as well as engineers. In this paper we present a new approach to modeling the swimming behavior of a jellyfish. Due to the axial symmetry of the creature we used a 2D cross-section for the calculation with the surface of the bell represented by two hemi-ellipsoidal curves. A simplified approach based on non-linear deformations of a geometric object is used to model the bell contraction-expansion cycle. We used a particle-gridless hybrid method for the analysis of incompressible flows, with averaging velocities field by the Shepard's method (partition of unity). To the best of our knowledge this is the first work where the optimal contraction and expansion parameters for the jellyfish movement were found by solving the optimization problem of maximizing the speed while minimizing the energy loss.

Keywords

Fluid dynamics, jellyfish, vortex, elasticity, optimization

1 INTRODUCTION

Jellyfish are the earliest known animals to use muscle power for swimming [DCC07]. They swim by contracting and expanding their mesogleal bells. The swimming muscles contract to expel a portion of water rearward out of the subumbrellar cavity, thus generating a thrust force to move the animal forward. The bell is refilled when it restores its shape after deformation it received during the thrust phase. The bell consists of a fiber-reinforced composite material called "mesoglea". The elastic characteristics of the mesogleal tissue were studied, for example, by Megill et al. [MGB05]

The contractile muscle fibers of the medusae are only one cell layer thick, so the forces that they can produce do not scale favorably with the increasing medusa size. For a medusa with the bell of diameter D, the mass of water that needs to be expelled from within the bell scales as D^3, while the muscle force only scales as D^1. Therefore the force required for jet propulsion increases with the animal size more rapidly than the available physiological force [DCC07]. Thus, the swimming performance may change dramatically with the increase of the medusan body size, and it is impossible to predict the optimal swimming parameters based on the geometric and kinematic similarity.

The physics of jellyfish swimming is not well understood. Existing animation techniques use combinations of sinusoidal curves to specify the deformations. However, it is important for animation to achieve a realistic movement depending on the size and shape of a bell. We assume that "realistic" also means "optimal", as the movements of the real jellyfish were "optimized" by the process of natural evolution, and we, therefore, would be able to find realistic movements for an artificial 3D model of jellyfish by means of artificial evolution. Other applications, such as computational biology, soft robotics and development of new propulsion techniques can benefit from development of a generalized model of jellyfish swimming.

In this paper we present a system for finding optimal swimming parameters for jellyfish models, based on our previous work where we studied vortex simulation for jellyfish [LS10]. The system consists of two main parts: simulated swimming and motion optimization. We introduce a simple technique based on radial basis functions (RBF) to model deformations of the jellyfish bell and a particle-gridless hybrid method for the analysis of incompressible flows. We modeled the interaction between the fluid particles and the surface of the bell in a form of elastic collision and reflection of the fluid particles off the boundary surface. The swimming efficiency was estimated for the bell and its particular movement specified by a set of control points. Genetic algorithms were used to find the optimal swimming pattern. To the best of our knowledge, this is the first work where the optimal swimming parameters for the jellyfish movement were found by solving the optimization problem. Throughout the paper we refer to two other paper concerning computational simulation

of jellyfish ([LM09] and [RM09]), but neither of those employs any numerical optimization.

The remainder of the paper is divided into 7 sections. In section 2 we discuss related work. We describe our approach in sections 3 to 5 and outline the algorithm in section 6. In section 7 we outline the specifics of our prototype implementation and report of the experimental results, and in section 8 we conclude and describe directions of future work.

2 RELATED WORK

2.1 Studies of real life jellyfish

Experimental studies, including dye injection, filming and analyzing the resulting flow, indicate that smaller prolate medusae create strong jets during their bell contraction stage. Bigger oblate medusae, however, produce substantially less distinct jets and broad vortices at the bell margins. A hypothesis proposed by Colin and Costello [CC02] [DCC07] [DCC03] [DCCG05] is that oblate species are using their bell's margins as "paddles", thus utilizing a paddling, or rowing, mode of swimming. According to the model presented by Dabiri et al. [DCC07], big oblate medusae are not capable of swimming via jet propulsion. There is, however, a study of McHenry and Jed [MJ03] which suggests that the jetting model still provides more accurate approximation of swimming in oblate jellyfish.

The flow generated by oblate medusa's pulsatile jets consists mostly of radially symmetric rotating currents called vortex rings. To better understand the vortex formation and their effect on swimming performance, numerous experimental studies of real live jellyfish were performed [CC02] [DCC07] [MJ03] [DG03] [DCC03] [DCCG05]. Researches using mechanical jet generators demonstrate that there is a physical limit – called the "vortex formation number" – for the maximum size of the vortex rings. Once this number is reached, no bigger vortex formation is possible, and the extra water creates a trailing current behind the vortex. The energy cost for generating this current is higher than that of creating the vortex ring, so it is optimal to generate the largest possible vortex without any trailing current [DCC03]. Both thrust and efficiency increase in direct proportion with vortex ring volume [DCCG05]. Lipinski and Mohseni [LM09] used digitized motions of two real hydromedusae to computationally simulate the flows. Their results confirm the hypothesis proposed by Colin and Costello and demonstrate that distinct type of jellyfish ("jetting" and "paddling") produce substantially different kinds of vortices.

2.2 Fluid-solid interaction

Müller et al. proposed a particle-based method for interaction of fluids with deformable solids [MST+04]. In their method they model the exchange of momentum between Lagrangian particle-based fluid model and solids represented by polygonal meshes with virtual boundary particles to model the solid-fluid interaction.

Lipinski and Mohseni [LM09] used digitized motions of two real hydromedusae to computationally simulate the flows. They used a new arbitrary Lagrangian-Eulerian method with mesh following the boundary between the fluid and the jellyfish body.

Yoon et al. presented a particle-gridless hybrid method for the analysis of incompressible flows [YKO99]. Their numerical scheme included Lagrangian and Eulerian phases. The moving-particle semi-implicit method (MPS) was used for the Lagrangian phase, and a convection scheme based on a flow directional local grid was developed for the Eulerian phase.

Chentanez et al. presented a method for simulating the two-way interaction between fluids and deformable solids [CGFO06]. The fluids were simulated using an incompressible Eulerian formulation where a linear pressure projection on the fluid velocities enforces mass conservation, whereas elastic solids were simulated using a semi-implicit integrator implemented as a linear operator applied to the forces acting on the nodes in Lagrangian formulation.

Hirato et al. proposed a method for generating animations of jellyfish with tentacles [HK03]. They used a simplified computational model based on the MPS method to simulate the fluid. Their work is mainly focused on visually plausible modeling of tentacles.

Rudolf and Mould created a system for physically-based animation of jellyfish [RM09]. Their approach may look very similar to ours, as they also exploited the radial symmetry, simulating only a 2D cross-section, and then creating a 3D bell for the visualization. The main difference between the approach proposed in [RM09] and the one discussed in this paper is that Rudolf and Mould did not employ any optimization, instead assigning a visually plausible set of parameters manually, by trial and error. They used a spring-mass system to represent the body of a jellyfish and a grid-based immersed boundary method for fluid-solid coupling. As they note in their work, there is still very little knowledge about physical properties of real jellyfish. Thus, we didn't feel necessary to employ something as complex as a spring-mass system, since the actual physical accuracy of the model would still be uncertain. Moreover, modeling a multi-layered structure of the jellyfish bell with only one layer of springs attached directly to the opposite sides of the bell does not look realistic. Some fugures from [RM09] demonstrate drastic change of both area and linear size of the umbrella cross-section during the contraction, something we failed to observe in real species, such as presented in experiments of Colin

and Costello [DCCG05]. Instead of a spring-mass system, we use a simpler approach, with the umbrella of the jellyfish represented in 2D as two spline curves, deformed by RBFs. Instead of a grid-based method, for fluid simulation we used a particle-based method [YKO99] with elastic collision and reflection of the fluid particles off the boundary surface to prevent fluid leaking across the boundary. Finally, [RM09] employs a very primitive visualization technique, an issue we were trying to address with a GPU-based parallel ray tracer, capable of representing transparency, reflectivity and venous structure.

2.3 Optimization

The problem was studied by many researchers from the computer graphics and animation community, but we have no room for the comprehensive referencing, so we will mention only a few we found most relevant to out work.

Sims was one of the pioneers of artificial evolution. In his work [Sim91] he used genetic algorithms to create evolving images, textures, animations and plants, represented by procedural geometry, with human aesthetical selection instead of a fitness function. In [Sim94] he used similar approach to artificially evolve both morphology and behavior of articulated (e. g. composed of rigid parts and connecting joints) creatures, which were evolved and trained to perform specific tasks, like walking, jumping, following a light source, competing for a ball with other creatures etc.

Terzopoulos et al. [TTG94] modeled artificial fish as NURBS and spring-mass systems, using simulated annealing to find efficient moving patterns. Based on simulated sensory input, their fish could learn complex group behaviour, such as schooling, mating etc.

Tan et al. [TGTL11] used covariance matrix adaptation to find optimal swimming motion for fish, frog, turtle and even some fictional creatures, represented as articulated bodies; however, they stated that their simulation method is unsuitable for soft body creatures, such as jellyfish.

The works, discussed in this section, inspired our attempt to create a combined approach suitable for modeling and optimizing jellyfish. We emphasize that our work unites two themes of different research history: generation of time-dependent shapes and estimation of dynamical characteristics of the generated models.

3 BELL SIMULATION

To simulate the bell contraction-expansion cycle we used a simplified approach based on non-linear deformations of a geometric object. Because the model of jellyfish has radial symmetry, we used a 2D model (cross-section) with the surface of the bell represented by two hemi-ellipsoidal curves – the upper and the lower. For our model we used a piece-wise linear approximation with the initial number of nodes equal 40. A space mapping technique based on RBFs (see [SS01], and references therein) was used for non-linear approximation of shape deformations in numerous applications. Space mapping in R^n defines a relationship between each pair of points in the original model and the model after geometric modification. Let an n-dimensional region $\Omega \subset R^n$ of an arbitrary configuration be given, and let Ω contain a set of arbitrary control points $\{q_i = (q_1^i, q_2^i, ..., q_n^i) : i = 1, 2, ..., N\}$for the non-deformed object, and $\{d_i = (d_1^i, d_2^i, ..., d_n^i) : i = 1, 2, ..., N\}$for the deformed object. By assumption, the points q_i and d_i are distinct and given on or near the surface of each of two objects. The goal of the construction of the deformed object is to find a smooth mapping function that approximately describes the spatial transformation. The inverse mapping function can be given in the form

$$q_i = f(d_i) + d_i, \tag{1}$$

where the components of the vector $f(d_i)$ are volume splines interpolating displacements of initial points q_i (see Appendix for the details).

4 FLUID-SOLID COUPLING

Using a grid-based approach for jellyfish is possible, but poses a number of problems. Using a regular grid, as in [TGTL11] for an elastic body with varying thickness will result either in a huge computational overkill (if the grid is dense enough to accomodate the thin edges), or in a poor accuracy of the computation (if the grid is more sparse). Using an irregular grid, as in [LM09] requires solving a mesh warping/re-meshing problem. Solving Navier-Stokes equations with moving boundary is a hard problem. For simplicity, we chose a particle-based method. Particle-based methods became a de-facto standard for a class of problems where high precision is not required. For modeling we used almost the same scheme as proposed by Yoon, Koshizuka and Oka [YKO99]. They proposed a particle-gridless hybrid method for the analysis of incompressible flows, where tracing of virtual moving particles is used instead of solving nonlinear equations of velocity field. A particle interacts with other particles according to a weight function $w(r)$, where r is the distance between two particles. The weight function used by Koshizuka et al. is

$$w(r) = \begin{cases} -(2r/r_e)^2 + 2 & (0 \le r < 0.5r_e) \\ (2r/r_e - 2)^2 & (0.5r_e \le r < r_e) \\ 0 & (r_e \le r) \end{cases} \tag{2}$$

Density for a particle is calculated as the sum of weights of its interactions with the other particles (all interaction happens only within the radius r_e):

$$\langle n \rangle_i = \sum_i w(|r_j - r_i|). \tag{3}$$

Note, that, unlike in the MPS method, the particle number density here is not required to be constant. A gradient vector between two particles i and j possessing scalar quantities ϕ_i and ϕ_j at coordinates r_i and r_j is equal to $(\phi_j - \phi_i)(r_j - r_i)/|r_j - r_i|^2$. The gradient vector at the particle i is given as the weighted average of these gradient vectors:

$$\langle \nabla \phi \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \left[\frac{\phi_j - \phi_i}{|r_j - r_i|^2} (r_j - r_i) w(|r_j - r_i|) \right], \quad (4)$$

where d is the number of space dimensions and n^0 is the particle number density.

Diffusion is modeled by distribution of a quantity from a particle to its neighbors using the weight function:

$$\langle \nabla^2 \phi \rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} [(\phi_j - \phi_i) w(r_j - r_i)], \quad (5)$$

where λ for a two-dimensional case with Equation (2) as the weight function is equal to $\frac{31}{140} r_e^2$. This model is conservative, because the quantity lost by the particle i is obtained by the particle j.

The continuity equation for incompressible fluid can be written as follows:

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot u) = 0. \quad (6)$$

The velocity divergence at the particle i is given by:

$$\langle \nabla \cdot u \rangle = \frac{d}{n^0} \sum_{j \neq i} \frac{(u_j - u_i) \cdot (r_j - r_i)}{|r_j - r_i|^2} w(|r_j - r_i|). \quad (7)$$

Then the pressure is calculated as:

$$\frac{u_i^{**} - u_i^*}{\Delta t} = -\frac{1}{\rho} \langle \nabla P^{n+1} \rangle_i, \quad (8)$$

$$\langle \nabla^2 P^{n+1} \rangle_i = \frac{\rho}{\Delta t} \langle \nabla \cdot u^* \rangle_i, \quad (9)$$

where u^* is the temporal velocity obtained from the explicit calculation and u_i^{**} is the new-time velocity. The left side of (9) is calculated using the Laplacian model (5). The right side is the velocity divergence, calculated by (7). We use variable r_e to avoid cases where some particles near the boundary will have very few neighbours to interact with. It gives a system of linear equations represented by an unsymmetric matrix, which is solved by an unsymmetric-pattern multifrontal method [Dav04]. Solving (9) may seem computationally expensive, but with jellyfish, the most important fluid-solid interaction often happens near the very thin edges of the bell, so calculating accurate pressure field is necessary.

Instead of using a higher-order gridless convection scheme as it was proposed by Yoon et al. [YKO99] to approximate flow directions, we applied averaging of the velocities field by a simple scheme, based on Shepard's method (partition of unity) [She68].

Boundary conditions are perhaps the most important factor influencing the accuracy of the flow computation. The manner in which the boundary conditions are imposed influences the convergent properties of the solution. Usually in particle-based methods boundary particles are used to approximate the no-penetration condition [MST+04] [PTB+03]. Repulsion and adhesion forces between the particles are used to simulate the no-penetration, no-slip and actio = reactio conditions on the boundary of the solid.

In our work contour points represent the geometry of the model and also define fluid boundaries. That is, the solution points are defined by the fluid particles and the particles located on the boundary of the bell. For each boundary particle we can calculate the boundary normal vector, pointing outwards, into the flow domain. For the no-slip condition, only the normal speed component of any boundary particle is used, while the tangential speed component is discarded. The no-penetration condition is modeled in a form of elastic collision and reflection of the fluid particles off the boundary surface. The motion of the bell was computed using only translational parts in y direction. One component of the force F on a rigid body is a derivative of linear momentum mv of the gravitational center. It is assumed that jellyfish body density is equal to the density of the water. Thus m is a volume occupied by the jellyfish.

The force F also invokes fluid and rigid body interaction. Points on the curve used to represent the bell can be considered as rigid particles. When the bell is deformed, distances between boundary particles may change, so we put a new set of boundary particles after each deformation, by evenly subdividing the curves. The strategy of using rigid particles we followed was first proposed in [CMT04]. The forces on rigid particles are computed by assuming the rigid body as a fluid. Therefore, for a particle i with the pressure p_i, mass density ρ_i and speed v_i, the force from the fluid acting on the node particle $f_i^{fluid} = f_i^{press} + f_i^{vis}$ is calculated by using the physical values of the neighbor particles as follows [DC96]:

$$f_i^{press} = -\sum_{i \neq j} (p_i + p_j)/2\bar{\rho}_j \nabla_i w_h^{ij} \quad (10)$$

$$f_i^{vis} = -\sum_{i \neq j} \Pi_{ij} \nabla_i w_h^{ij} \quad (11)$$

where

$$\Pi_{ij} = \begin{cases} -(c\mu_{ij} + 2\mu_{ij}^2)/\bar{\rho}_j & \mu_{ij} < 0 \\ 0 & \mu_{ij} \geq 0 \end{cases} \quad (12)$$

$\mu_{ij} = e_r v_{ij} r_{ij}/(r_{ij}^2 + 0.01 e_r^2)$, $\bar{\rho}_j = 0.5(\rho_i + \rho_j)$, $r_{ij} = r_i - r_j$, $v_{ij} = v_i - v_j$

5 OPTIMIZATION

In techniques based on the error functional minimization it may become necessary to solve highly non-linear

problems. Minimization by standard techniques requires high computational effort. Minimization of a simplified functional, for example a quadratic one, is reduced to solving a simple system of linear equations. However, it leads to iterative minimization that depends on a sufficiently good initial guess. It seems to us that an attractive way of attacking this problem is to use optimization techniques based on genetic algorithms, proposed by Mahfoud and Goldberg in [MG92]. In this work we used an algorithm with simulated annealing type selection.

The application of the genetic algorithm starts with initially selecting a set of M variable control points $\{d_i = (d_{1i}, d_{2i}, d_{3i}, ...) : i = 1, 2, ..., M\}$ for the definition of the space transformation generating the deformed object. Actually, the user defines points q_i on the initial image of the bell in its rest state with corresponding points d_i on the model of fully contracted bell (Fig. 1). The collection of coordinates d_i and the contraction time t_{cont} define a creature. The algorithm begins by randomly distorting the initial creature and generates s creatures, which form the initial population. Now, the genetic algorithm with sequential simulated annealing is applied to this initial population to minimize the fitness function.

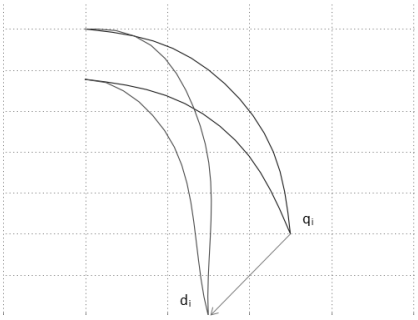


Figure 1: An example of a creature: the right half of the bell cross-section in two states (initial and deformed) and the deformation vector

The spline $f(P)$, determined by the set of N variable control points d_i which constitute a creature, used for global space mapping, provides a minimization of quantity $h^t A^{-1} h$, that is called "bending energy". 8 points belonging to the border of the bounding box and two additional points in the center of the bell ($x = 0$) on the upper and lower curve are used as anchor points. k destination points define general deformation of the jellyfish bell. A^{-1} is the bending energy matrix, which is the inverse $N \times N$ upper left submatrix of T, and h_i are so-called heights and $N = 10 + k$. Space transformation h_i is the difference between the coordinates of the initial and destination point placements as shown in Fig. 1. The bending energy of a general transformation is the sum $x^t A^{-1} x + y^t A^{-1} y$ of the bending energy of its horizontal x-components, modeled as a "vertical" plate, and the bending energy of its vertical y-component, modeled similarly as a "vertical" plate.

In our simulation we used the "economy" principle: the jellyfish is striving to reach maximum speed with minimum deformation of the bell. Thus, one of the fitness function component is the bending energy E^b. In the numerical analysis we also measured two quantities characterizing jellyfish locomotion, i. e. distance D passed by a body which is defined by swimming speed v and energy loss E^l. Energy loss is assumed to be equal to surrounding water energy. Following the "economy" principle, we define the fitness function as follows:

$$Fitness = w_v \cdot D - w_b \cdot E^b - w_e \cdot E^l \qquad (13)$$

where w_b is the weight for RBF energy, w_e is the weight for kinetic energy of a particle, w_d is the weight for the object velocity. These parameters are set by the user to choose a mode of movement.

6 ALGORITHM

Initially, the 2D contour of the bell cross-section is specified as an array of points. Deformations are assigned to the bell margin points. Particles are placed at a regular interval (on a regular grid) inside the bounding box, except the inner area of the bell. Then, the following steps are performed iteratively for each step of the contraction/expansion cycle:

1. The averaged density is calculated for every particle. A ball is generated for every particle, and the density is defined as the volume of the ball divided by the number of particles inside the ball. The ball diameter is not constant, and is adjusted so that all balls contain roughly similar number of particles.

2. The bell margin points are moved by a step along the deformation vectors. For the rest of the points their displacement vectors are calculated using RBFs.

3. A cardinal spline is fit through the displaced boundary points. Because some segments may become too long, we discard the boundary points and insert them again by subdividing the spline curve evenly, so that all the distances between neighboring points are mostly equal.

4. A Poisson equation in a matrix form (unsymmetric, about 10000 linear equations) is solved, giving new values of pressure for each particle.

5. Gradient vectors are calculated applying equation (4). For every particle, the speed vector is calculated, and the particle is then moved along the vector by the time step Δt.

6. New pressure values for the displaced particles are interpolated back to the nodes of the regular grid.

Number of sub-steps	Time (sec)	Path (m)
6	7	0.02
10	13.14	0.032
14	18	0.031
18	23.99	0.033
20	26.6	0.038
40	25.39	0.039

Table 1: Performance results

7. Distance passed and energy lost at this step are calculated for the creature.

At the end of a swimming cycle, we have a fitness for the creature according to (13). A population of 10 such creatures is evolved until convergence within 10%. The best creature is then selected as "optimal".

For animation, we created a 3D model out of the 2D contour, and then visualized by ray tracing. At first, the mesh was created by rotational extrusion and tessellation of the original 2D contour. Finally, we used Blender 3D modeling suite to create a roughly similar 3D model with some embellishments, such as inner "veins", inward-oriented "velum" and several tentacles. Rotation of the original 2D contour was still used to put anchor points at some interval around the bell. The anchor points were used as a "skeleton" for RBF-based deformations of the entire bell model in 3D, including the veins, at each animation step. Some random perturbations were added to the anchor points to make our jellyfish look less artificial (Fig. 2). The tentacles were modeled as soft cloth with one side attached to the bell and deformed separately. The cloth structure was represented by a triangular mesh, with nodes affected by the water flow.

7 RESULTS AND DISCUSSION

We implemented this method and used it to find the optimal swimming parameters for a simple oblate jellyfish similar to Aurelia aurita [DCCG05]. The program was written in C++ and CUDA C. UMFPACK library [Dav04] was used for solving large sparse systems of linear equations. The algorithm terminates when a satisfactory fitness level has been reached for the population. In practice it happens when the number of generations is approximately 30. As Table 1 shows, the path differs at the 3rd decimal digit. In our implementation we typically used 14 substeps with the calculation time of about 10 minutes on a single core of an Intel Core 2 Quad computer. The size of the simulation area is 20x20 cm, with the resoultion of 100x100 particles. The bell diameter is 10 cm.

In the particular case (see Fig. 2) weights (lazy mode) $w_b = 0.3$, $w_e = 0.3$ and $w_d = 0.4$ were used. The vortices produced by the simulation (Fig. 3) were similar to those observed for real jellyfish, showing that application of the no-slip condition looks reasonable.

A GPU-based parallel ray tracing system was developed for the visualization (Fig. 2). We used recursive ray tracing to visualize the bell as a transparent object with refraction and reflection. Each primary ray hitting the bell was spawning several secondary rays, so the animation performance varied depending on the number of primary rays hitting the bell, and, thus, on the distance from the camera to the jellyfish and on the viewing angle. Our ray tracer produced 20-30 frames per second on a GeForce GT 540M GPU. See the supplementary video for an example of animation. We must add, that experience in areas such as 3D art and texture painting would add significantly to the observed realism of the animation, but that is beyond the scope of this work.

To validate our results, we compared them with experimental data received by Colin and Costello [CC02] [DCCG05]. The distance passed by our model during one full contraction-expansion cycle (which is roughly equal to the bell radius) and the resulting water flows seem to be in agreement with their data.

8 LIMITATIONS AND FUTURE WORK

We employed our simulation software to find the optimal movement for a very simple 2D model, swimming straight ahead. A more thorough validation of our technique, using a variety of sizes and shapes, as well as robustness and sensitivity studies are required and are subjects of future work. A few control points were used to specify bell contraction, and the movements of the bell margins were set by only two axially-symmetric vectors. Real jellyfish have no brain or eyes (although some of them have photosensitive spots on their bells) and do not deliberately choose any complex swimming trajectory, so we think our simplification has no big impact on the results veracity for jellyfish. However, following complex paths would be crucial for jellyfish-like robots. To simulate such behavior, it may be necessary to perform the simulation in 3D and with larger number of (possibly asymmetric) control points.

We did not, either, take into account jellyfish feeding behavior and tentacles. Real jellyfish have an oral opening inside the bell. Some of them also have numerous tentacles spread in the water. The tentacles add drag force and decrease swimming performance, but are used to catch prey. Jellyfish create water flows to carry their prey through the tentacles or into the oral cavity itself. Modeling such behavior is important for computational biology. Additional parameters for it can be incorporated into the fitness function.

Another possible future work would be optimization of the shape itself, e. g. finding both optimal movement and optimal shape, satisfying constraints imposed by a 3D designer.

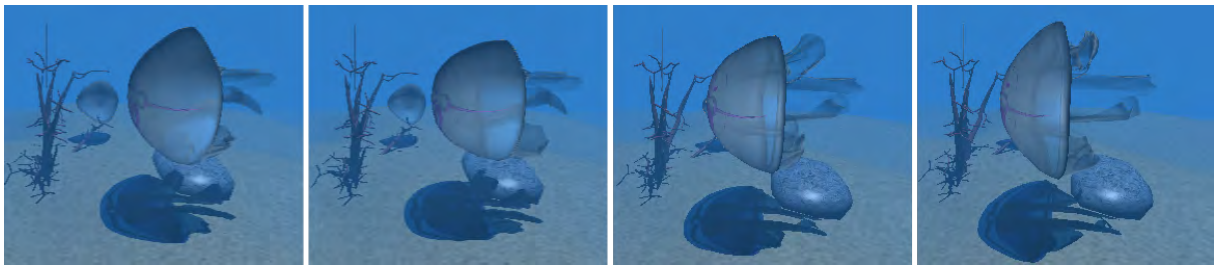


Figure 2: Animation sequence of two contraction steps (left) and two expansion steps (right) for a jellyfish with floating tentacles and small additional deformations applied to the bell.

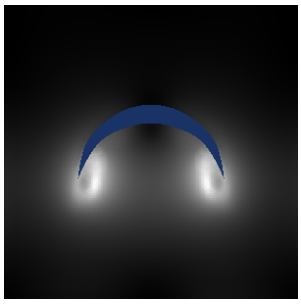


Figure 3: Vortex formation

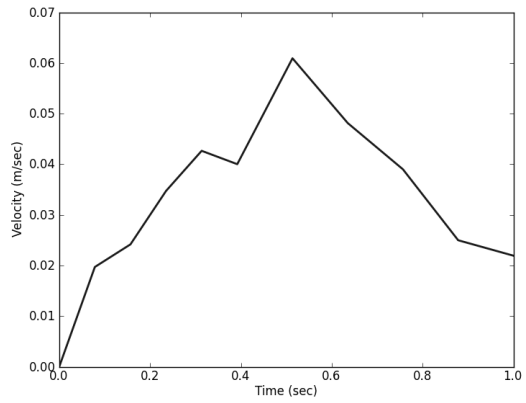


Figure 4: Velocity change for a jellyfish during the initial swimming cycle (10.00 cm diameter, 0.39 sec contraction time, 0.61 sec expansion time)

9 ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful comments.

10 APPENDIX

We consider a mapping function as a thin-plate interpolation. For an arbitrary area Ω, the thin-plate interpolation is a variational solution that defines a linear operator T when the following minimum condition is used:

$$\int_{\Omega} \sum_{|\alpha|=m} m!/\alpha!(D^{\alpha} f)^2 d\Omega \rightarrow min, \tag{14}$$

where m is a parameter of the variational function and α is a multi-index. It is equivalent to using the RBFs $\phi(r) = rlog(r)$ or r^3 for $m = 2$ and 3 respectively, where r is the Euclidean distance between two points.

The volume spline $f(P)$ having values h_i at N points P_i is the function

$$f(P) = \sum_{j=1}^{N} \lambda_j \phi(|P - P_j|) + p(P), \tag{15}$$

where $p = v_0 + v_1 x + v_2 y + v_3 z$ is a degree-one polynomial. To solve for the weights λ_j we have to satisfy the constraints h_i by substituting the right part of Equation (15), which gives

$$h_i = \sum_{j=1}^{N} \lambda_j \phi(|P_i - P_j|) + p(P_i). \tag{16}$$

λ and v are the coefficients that satisfy a linear system $Tx = b$, where

$$\begin{aligned} T &= \begin{bmatrix} A & B^T \\ B & D \end{bmatrix}, \\ x &= [\lambda_1, \lambda_2, ..., \lambda_N, v_0, ..., v3]^T, \\ b &= [h_1, h_2, ..., h_N, 0, 0, ..., 0]^T \end{aligned} \tag{17}$$

For 2D and 3D cases we call $f(P)$ a volume spline.

REFERENCES

[CC02] S. P. Colin and J. H. Costello. Morphology, swimming performance and propulsive mode of six co-occuring hydromedusae. *The Journal of Experimental Biology*, 206:427–437, 2002.

[CGFO06] N. Chentanez, T. G. Goktekin, B. E. Feldman, and J. F. O'Brien. Simultaneous coupling of fluids and deformable bodies. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, pages 83–89, 2006.

[CMT04] M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Transactions on Graphics*, volume 23, pages 377–384, 2004.

[Dav04] T. A. Davis. Umfpack, an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, June 2004.

[DC96] M. Desburn and M. P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. *Computer animation and simulation*, pages 61–67, 1996.

[DCC03] J. O. Dabiri, S. P. Colin, and J. H. Costello. Fast-swimming hydromedusae exploit velar kinematics to form an optimal vortex wake. *The Journal of Experimental Biology*, 206:3675–3680, 2003.

[DCC07] J. O. Dabiri, S. P. Colin, and J. H. Costello. Morphological diversity of medusan lineages constrained by animal-fluid interactions. *The Journal of Experimental Biology*, 210:1868–1873, 2007.

[DCCG05] J. O. Dabiri, S. P. Colin, J. H. Costello, and M. Gharib. Flow patterns generated by oblate medusan jellyfish: field measurements and laboratory analyses. *The Journal of Experimental Biology*, 208:1257–1265, 2005.

[DG03] J. O. Dabiri and M. Gharib. Sensitivity analysis of kinematic approximations in dynamic medusan swimming models. *The Journal of Experimental Biology*, 206:3675–3680, 2003.

[HK03] J. Hirato and Y. Kawaguchi. Calculation model of jellyfish for simulating the propulsive motion and the pulsation of the tentacles. *18th International Conference on Artificial Reality and Telexistence*, 2003.

[LM09] D. Lipinski and K. Mohseni. Flow structures and fluid transport for the hydromedusae Sarsia tubulosa and Aequorea victoria. *The Journal of Experimental Biology*, 212:2436–2447, 2009.

[LS10] V. Lazunin and V. Savchenko. Vortices formation for medusa-like objects. *Proceedings of Fifth European Conference on Fluid Dynamics (ECCOMAS CFD 2010)*, June 2010.

[MG92] S. W. Mahfoud and D. E. Goldberg. A genetic algorithm for parallel simulated annealing. *Parallel problem solving from nature*, 2:301–310, 1992.

[MGB05] W. M. Megill, J. M. Gosline, and R. W. Blake. The modulus of elasticity of fibrillin-containing elastic fibres in the mesoglea of the hydromedusa polyorchis penicillatus. *The Journal of Experimental Biology*, 208:3819–3834, 2005.

[MJ03] M. J. McHenry and J. Jed. The ontogenetic scaling of hydrodynamics and swimming performance in jellyfish (aurelia aurita). *The Journal of Experimental Biology*, 206:4125–4137, 2003.

[MST+04] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. Interaction of fluids with deformable solids. *Computer Animation and Virtual Worlds*, 15:159–171, 2004.

[PTB+03] S. Premože, T. Tasdizen, J. Bigler, A. Lefohn, and R.T. Whitaker. Particle-based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410, 2003.

[RM09] D. Rudolf and D. Mould. Interactive jellyfish animation using simulation. *International Conference on Computer Graphics Theory and Applications (GRAPP)*, pages 241–248, 2009.

[She68] D. Shepard. A two-dimensional interpolation function for irregularly spaced data. *Proceedings of the 23th Nat. Conf. of the ACM*, pages 517–523, 1968.

[Sim91] K. Sims. Artificial evolution for computer graphics. *Computer graphics*, pages 319–328. ACM SIGGRAPH, July 1991.

[Sim94] K. Sims. Evolving virtual creatures. *Computer graphics*, pages 15–22. ACM SIGGRAPH, July 1994.

[SS01] V. Savchenko and L. Schmitt. Reconstructing occlusal surfaces of teeth using genetic algorithm with simulated annealing type selection. *6th ACM Symposium on Solid Modeling and Applications*, pages 39–46, June 2001.

[TGTL11] J. Tan, Y. Gu, G. Turk, and C. K. Liu. Articulated swimming creatures. *Computer graphics*, volume 30. ACM SIGGRAPH, July 2011.

[TTG94] D. Terzopoulos, X. Tu, and R. Grzeszczuk. Artificial fishes: autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, 1(4):327–351, 1994.

[YKO99] H. Y. Yoon, S. Koshizuka, and Y. Oka. A particle-gridless hybrid method for incompressible flows. *International Journal for Numerical Methods in Fluids*, 30:407–424, 1999.

A Comprehensive Taxonomy for Three-dimensional Displays†

Waldir Pimenta
Departamento de Informática
Universidade do Minho
Braga, Portugal
wpimenta@di.uminho.pt

Luís Paulo Santos
Departamento de Informática
Universidade do Minho
Braga, Portugal
psantos@di.uminho.pt

ABSTRACT

Even though three-dimensional (3D) displays have been introduced in relatively recent times in the context of display technology, they have undergone a rapid evolution, to the point that a plethora of equipment able to reproduce dynamic three-dimensional scenes in real time is now becoming commonplace in the consumer market.
This paper's main contributions are (1) a clear definition of a 3D display, based on the visual depth cues supported, and (2) a hierarchical taxonomy of classes and subclasses of 3D displays, based on a set of properties that allows an unambiguous and systematic classification scheme for three-dimensional displays.
Five main types of 3D displays are thus defined –two of those new–, aiming to provide a taxonomy that is largely backwards-compatible, but that also clarifies prior inconsistencies in the literature. This well-defined outline should also enable exploration of the 3D display space and devising of new 3D display systems.

Keywords

three-dimensional displays, depth cues, 3D vision, survey, taxonomy

1. INTRODUCTION

The human ability for abstraction, and the strong dependence on visual information in the human brain's perception of the external world, have led to the emergence of visual representations of objects, scenery and concepts, since pre-historical times. Throughout the centuries, many techniques have been developed to increase the realism of these copies.

Recent years have revealed a focusing of these efforts in devising ways to realistically recreate the sensation of depth, or three-dimensionality, of the depicted scenes. 3D displays thus emerged as an active area of research and development.

Despite this being a relatively recent field, many different approaches for 3D displays have been already proposed and implemented, and new ones surface with some regularity. Moreover, these implementations provide different sets of approximations for the depth cues that our visual system uses to perceive the three-dimensionality of a scene.

This profusion of implementations has plagued attempts to define a nomenclature system for 3D displays. While a few comprehensive classification schemes have been proposed, most based on or compatible with Okoshi's seminal work [Oko76], several are now obsolete, while recent attempts tend to be specific to a subset of displays [MK94, Hal97, BS00, Dod05, GW07, UCES11].

Mostly what is seen are overview sections in publications that on the one hand assume implicit definitions of 3D perception and 3D displays, and on the other hand frequently avoid taking a stance (or do so inconsistently) in undecided issues emerging from partially incompatible previous classifications, such as the placement of holographic technology [Fav05] or integral imaging [DM03]. A definitive, exhaustive and unambiguous categorization system for 3D displays has thus been lacking in the literature [CNH+07, p.1], which hinders the classification and evaluation of different implementations, especially hybrid ones.

The approach presented in this paper focuses in the formalization of the properties of each category of 3D displays, to provide a stable system for classifying existing or new implementations. Specifically, the definition and categorization of 3D displays is based in their fundamental properties, rather than in implementation details, as is the case with most current classifications.

†*This work has been funded by the Portuguese agency FCT – Fundação para a Ciência e a Tecnologia, through the grant SFRH/BD/74970/2010.*

As a necessary foundation for this taxonomy, Section 2 presents a general overview of the depth cues used by the human visual system to perceive three-dimensionality. With this knowledge, we can then, in Section 3, determine the specific subset of these that clearly mark the frontier between 2D and 3D displays, and define basic properties of 3D displays. Section 4 then delves into the 3D display realm, defining a hierarchy of types and subtypes for 3D displays, based primarily on the depth cues they implement.

By employing a systematic approach, we expect the outcome to be a logical, well-structured and extensible taxonomy that will facilitate comparison of different approaches, and the evaluation of appropriate techniques for a given application. The Conclusion assesses the degree to which this objective was fulfilled, and illuminates what further work is to be performed to complement the proposed taxonomy.

2. VISUAL CUES TO THREE-DIMENSIONALITY

The origins of the Human species, as primates living and moving in trees, and later as hunter-gatherers, contributed significantly to make perception of depth a very important feature of our vision. Developments in art and research in optics and display technology have revealed some of the cues that our visual system uses to interpret the location of objects. These hints, known as **depth cues**, can be divided into two main groups: *psychological* cues, which depend on acquired knowledge of the visual aspect of familiar objects, and *physiological* cues, which manifest through the anatomy of our visual system [Oko76].

The main psychological depth cues are:

Occlusion. The overlap of some objects by others that are closer to us. This is one of the most fundamental ways we perceive depth on a scene.

Linear perspective. Given prior knowledge of common shapes and/or sizes of objects, we interpret perceived distortions in their shape (parts farther away from us appear smaller), differences in size between them, and variation of their angular size (how much of our visual field they cover) as indicators to their location in three-dimensional space.

Atmospheric perspective. Commonly known as "distance fog", it refers to the fading in contrast and detail, and shift to bluish colors, of objects located at a great distance. This happens because the light we get from them had to travel an increased distance through air and thus underwent more scattering from the atmospheric gases and particles.

Shading and **shadow projection.** Effects caused by the relationship between objects and light sources. The distribution of brightness and color in an object's surface provides information about (among other things) its shape and position relative to the light sources that illuminate it. Also, the location, format and darkness of shadows projected into the object (due to parts of it or other objects obscuring the light) and into its vicinity allow us to interpret its 3D form and relative position to other objects and/or the environment.

The above are all static cues. There are two more psychological cues, which are dynamic; that is, they manifest when there is movement either of the observer or of the observed object (or both):

Motion parallax. Relative changes in perceived position between two objects when we move. For example, during a car trip a tree seems to be "travelling past us" faster than the distant mountains.

The kinetic depth effect. Changes in the appearance of an object due to its own motion. For example, when a spherical object –say, a football– is uniformly illuminated so that no shadows give away its round shape, a slow rotation around itself is sufficient for our visual system to infer that it is a solid body and not a flat disk facing us, due to the relative motions of features in its surface.

The physiological depth cues consist of:

Binocular disparity (or **stereo parallax**)[1]. Differences in images received by each eye, commonly called **stereoscopy**[2]. Studies indicate [Oko76] that for a moderate viewing distance, binocular disparity is the dominant depth cue to produce depth sensation, through a process called *stereopsis*, which is the effort made by the brain to fuse the images together into a 3D perception of the scene. This fusion effort is always necessary because convergence of the eyes can only produce a perfect match for a limited subset of the points from the images, due to projection geometry constraints.

Convergence. When both eyes rotate inwards to aim at the object of interest, thus aligning the different images they receive, so they can be more effectively combined by the brain. As with accommodation, this rotation manifests itself with greater amplitude when differences in distance occur closer to the eye, so it is also a cue that is more strongly perceived for nearby objects (less

[1] "Binocular" comes from the Latin *bini* (pair) + *oculus* (eye). "Stereo" comes from the Greek *stereós* (solid).

[2] It's been known since as early as 300 B.C. that depth perception in human vision is related to the fact that we have two eyes, in separate physical locations, which collect different simultaneous perspectives of the same object [EucBC].

than 10m, according to [Wid01]). If they are close enough, one can clearly feel the eyes "crossing" so that they can keep aiming at the same point.

Accommodation. The effort made by the muscles in the eye that control the shape of its lens in order to bring the image into focus in the retina. Even though we usually do not consciously control these actions, our brain uses this muscular contraction information as an indicator of the distance of objects we are observing. Since the focusing effort varies much more for distance changes near the eye, the effect is particularly notable for nearby objects (less than 2m, according to [MZ92]).

The depth cues described above are summarized in Table 1.

The Accommodation-Convergence Mismatch

The fact that most visual representational media are unable to implement all depth cues –especially the physiological ones–, does not pose a serious problem, either because the scenes represented are meant to take place (or be viewed from) a distance where the physiological cues aren't relevant [Oko76, p.39], or because we can cognitively ignore the mismatch in psychological vs. physiological depth cues, as our abstraction ability allows us to understand their purported three-dimensionality regardless.

However, a mismatch *among* the physiological cues is less tolerable. This mismatch is common in current 3D displays, because every display that provides stereoscopy (one view for each eye) is theoretically able to implement proper convergence cues for each object in the scene depending on their location. But accommodation (provided by the ability to make the light rays diverge not from the screen, but from the virtual positions of the scene objects) is much harder to achieve; therefore, most of these displays end up forcing the eye to always focus at the screen to get a sharp image, which conflicts with the cues of convergence and stereopsis. The resulting phenomenon is called the *accommodation-convergence mismatch*.

This mismatch is more serious than the aforementioned one, because providing the brain with conflicting physical signals causes discomfort, the same way mismatch between visual and vestibular (from the balance system in the inner ear) perception of movement causes motion sickness. The consequences may include headaches, fatigue or disequilibrium, preventing continued use of these displays. This, of course, in addition to the reduction it causes in the realism of the 3D visualization, which might become uninteresting or even visually confusing [Hal97].

Table 1: Summary of visual depth cues for three-dimensional vision

	static	**dynamic**
psychological	occlusion (overlap); linear perspective; atmospheric perspective (distance fog); shading and shadows.	motion parallax; kinetic depth effect.
physiological	accommodation (focus); binocular disparity (stereoscopy); convergence.	

3. DEFINITION OF A 3D DISPLAY

Before defining what a 3D display is, it is necessary to clarify what is meant by "display". As a word with multiple meanings, we will assume the context of visual perception and the word's usage as a concrete noun (i.e., the name of a thing). As such, the definition adopted will be "a visual output device for the presentation of images".

It's worth pointing out that the word "images" is in plural, because we will consider only display media that don't produce permanent records, but instead are mutable, or rewritable, by comprising reconfigurable active elements, such as pixels, voxels[3] or catoms[4] – in other words, *electronic visual displays*. This, as [Oko76] pointed out, effectively excludes static visual representations such as paintings, photographs, sculptures, and even classical (static) holograms, for they are not displays in the sense adopted above, but merely the physical embodiment of a specific image. These will therefore be left out of this taxonomy. Nevertheless, all the principles behind them are present in the displays we consider, the only difference being the adoption of a rewritable medium.[5]

With the clarification of what constitutes a display device, we can now approach the question of what makes a display three-dimensional. Firstly, we must acknowledge that the line separating 3D displays from 2D displays is not always clearly defined, despite what the dichotomic "2D/3D" nomenclature seems to suggest. This fuzziness occurs because, on the one hand, the psychological 3D depth cues can, in fact, be reproduced in media traditionally considered as 2D; and on the other hand, many displays deemed three-dimensional are actually flat screens, which means that the images are emitted from a two-dimensional surface.

[3] A *portmanteau* of the expression "volumetric pixels".

[4] In the (still theoretical) field of claytronics –dynamic sculptures made of microscopic robots–, "catom" is a combination of the words "claytronic atoms" [GCM05].

[5] For instance, when we mention holography, or stereoscopic displays, we will be referring to their electronic counterparts.

With these limitations in mind, **we define 3D displays as visual output devices that evoke at least one of the physiological depth cues** (stereoscopy, accommodation and convergence) – besides, naturally, the psychological cues enabled by the specific display technology used. This definition ensures that the 3D perception is truly engaged in a natural way, and not by ignoring the apparent flatness of the scene, as happens with displays based only in psychological depth cues.

4. PROPOSED TAXONOMY FOR 3D IMAGING TECHNIQUES

To define a basis for the proposed taxonomy, we will apply two general criteria as orthogonal axes of categorization. We'll demonstrate that by intersecting these two basic properties, it is possible to establish a well-grounded, formally-defined taxonomy that largely validates current consensus but also clarifies conflicting definitions.

The first axis is the **number of views supported** by the display. The reasoning behind this is that most of the depth cues for 3D perception (occlusion, motion parallax, convergence, stereopsis, etc.) are dependent on the angle from which the observer views the scene. 3D displays will employ different methods to emulate this viewpoint-dependent variation of the light field.

One such method consists simply in producing two views and ensuring that each is only seen by the appropriate eye of the observer. Another approach employs displays that are able to project multiple views into different directions. This is implemented by segmenting the image into as many perspectives as desired, multiplexing them into the display, and using a filtering mechanism to direct each view to the corresponding direction. Finally, a third type comprises displays that can generate or approximate a continuous wavefront of light that propagates as coming from the actual 3D position of the virtual object, rather than dispersing from its projection in the display surface.[6]

Throughout the years, as 3D displays advanced past the two-views (binocular) approach, the word "stereoscopic" has gradually expanded its range to become largely synonymous with three-dimensional vision (and rightly so), and is thus routinely applied to displays of all of these types. Therefore, in the spirit of unambiguity, the three meta-categories described above will be named *"duoscopic"*, *"multiscopic"* and *"omniscopic"*, respectively.[7]

[6]Head tracking by itself only implements monocular directional variation; thus, it doesn't constitute a 3D display as defined above.

[7]Prior attempts to define the difference between these types of displays have entailed the use of the terms "stereograms" and "panoramagrams" [Oko76, Hal97], but the distinction hasn't been widely adopted in the literature, and even less in the industry.

The other main axis we'll use to map the 3D displays space is the **effective shape of the display medium** itself, which can be *"flat"* or *"deep"*. This doesn't depend strictly on the shape of the display surface, but rather on the effective volume it occupies while displaying the 3D image. The flat displays can be compared to a window, a planar surface which provides different perspectives as one moves around, but limits the scene at its boundaries. For the deep displays, there is a volume of space occupied by the display medium (either permanently or due to moving elements) and the virtual object is displayed inside the volume, also not able to exist outside the volume's boundaries as they are perceived by the observer. We can say that one looks *through* flat displays as if through a window, and looks *into* deep displays as if they were a crystal ball.

Aside: the projection constraint

The boundary limitation of both the flat and the deep displays are manifestations of the *"projection constraint"*.[8] Countering this effect may be done by increasing the absolute size of the display (for example, a cinema screen), shaping it in order to surround the viewer (as is done in the CAVE virtual reality environment), or increasing its relative size by bringing it closer to the observer (the technique used by virtual reality glasses).

These two criteria allow us to effectively separate the displays into five main categories, most of which are already well-established in the literature. Table 2 summarizes this division.

It might be noticed that two of those terms are not common in most taxonomies, namely "virtual volume displays" and "multi-directional displays". They are, in fact, key components of this taxonomy, in that they clarify the classification of techniques for which past works have not been able to agree on a category. Other categories, however, were included with their currently *de facto* standard names, in order to prevent excessive disruption and preserve as much backwards-compatibility as sustainable without breaking the consistency of the proposed framework.

Table 2: Proposed Taxonomy

		display shape	
		flat	*deep*
# views	*duoscopic*	stereoscopic	
	multiscopic	autostereoscopic	multi-directional
	omniscopic	virtual volume	volumetric

[8][Hal97] describes the projection constraint by stating that "a display medium or element must exist in the line of sight between the viewer and all parts of the [visible] image."

In the following subsections we will complete the definition of these five groups by specifying their main properties and, where applicable, defining relevant subcategories inside them.

Flat 3D displays

Flat-type, screen-based 3D displays are the most popular kind of 3D displays used currently, with commercial use now common in movie theaters and domestic entertainment devices. They work mostly by providing stereoscopy (different images for each eye), which, as mentioned in Section 2, is the main depth cue for 3D vision at moderate distances.

These displays can be further divided in three main groups: **stereoscopic** devices, which work in conjunction with glasses to provide two distinct views; **autostereoscopic** screens, which can generate multiple views without requiring any headgear; and **virtual volume** displays, which recreate the 3D wavefront as if propagating from the actual location of the 3D image – the most notable example being the hologram.

4.1.1 Stereoscopic Displays

Stereoscopic 3D displays can display one image to each eye in two ways: either by combining (i.e, multiplexing) two separate streams of images in one device, and filtering them with special glasses, or by using separate display devices for each eye.

Glasses-based stereoscopic displays can be implemented through three filtering techniques [Ben00]:

Wavelength multiplexing. Separating the left-eye and right-eye images in different colors, the most well-known example of which is the anaglyph, with its characteristic "red-green" glasses;

Temporal multiplexing. Using shutter glasses synchronized with the screen and a doubled frame-rate that displays the images for the left and right eye alternatively;

Polarization multiplexing. Achieved by emitting images for each eye with different light polarizations (direction of wave oscillation), and filtering them with polarized-filter glasses.

The stereoscopic displays that use separate screens for each eye are usually called **head-mounted displays (HMDs)**. This name is justified because the whole display system is head-mounted, rather than only the filtering mechanism.

HMDs include mostly devices such as virtual reality (VR) or augmented reality (AR) glasses, but also comprise techniques still largely embryonic, such as retinal projection, contact lens displays and brain-computer interfaces.

As previously mentioned, HMDs can overcome the projection constraint by displaying the image closer to the eye, thus increasing its relative size and coverage of the visual field.

There are two key characteristics of stereoscopic displays that separate them from other 3D vision techniques: (1) they require either the whole display system or the filtering mechanism to be fixed regarding the eyes, which in most cases implies some sort of headgear, thus being potentially invasive to varied degrees (ranging from light and inexpensive filtering glasses to surgery-requiring neural implants), and (2) because they only present two views, they only support a single user/perspective.[9]

Motion parallax is not natively supported by stereoscopic displays, but they can be enhanced to support it by employing head tracking [Dod05].

4.1.2 Autostereoscopic Displays

Autostereoscopic screens are usually implemented using two techniques:

Parallax barriers, which work by sequentially interlacing the images for each perspective in vertical strips, and employing a fence-like barrier that restricts the light from each strip to propagate only in its corresponding direction.

Lenticular displays, which do this filtering by using an array of lenses that direct each part of the image to the correct direction. These lenses are usually cilindrical, providing only horizontal parallax, but spherical lenslets have been proposed to overcome this limitation, resulting in what is called an "integral imaging" device.

Autostereoscopic screens exploit the fact that the eyes occupy different points in space to provide stereoscopy. In other words, they employ direction-multiplex to channel information of the left and right views into appropriate eyes [DM03].

These direction multiplexing techniques can be generalized to produce more than two views, which enables motion parallax, and consequently the ability to support multiple observers with a single display, without any headgear. However, undesired optical distortions caused by too small lenses or barriers limits the number of possible views. The motion parallax supported is thus markedly non-continuous, which reduces the realism of the 3D effect [Hal97].

[9] It is possible, using HMDs, to implement multi-user applications by having each user wear their own device, and keeping all of them synchronized, but this is obviously a costly and technically challenging approach.

Anisotropic diffusers (surfaces that scatter light in very narrow horizontal directions) have been presented as a potential solution to such limits [UCES11]. It's been reported[Tak06] that with enough angular resolution, such displays could even create accomodation responses in the eye. Therefore, by sufficiently approximating (in the assigned visualization area) the continouous wavefront that a real object would create, they could be considered omniscopic instead.

4.1.3 Virtual Volume Displays

Virtual volume displays, as the name says, are able to generate the sensation of depth by placing virtual images in 3D space, without having to physically span the imaging volume [Hal97]. Since each point of the image is optically located at the correct depth, these displays are able to provide proper accommodation. This can be implemented either by adaptive optics, or through the holographic technique.

Adaptive optics employ dynamic optical systems that can change their focusing power. These can be deformable (varifocal) membrane mirrors, or "liquid" lenses, usually produced through an effect called "electrowetting". They are similar to the old illusion called *Pepper's ghost*, which consists in a semi-transparent mirror that superimposes a reflection (of a real object, or a verisimilar 2D projection) over the background scene, producing a ghostly image of the object, and which still finds modern use in many theme parks and live shows.

In displays based on adaptive optics, the flexible optical element will reflect or transmit a static screen that displays a sequence of depth slices, synchronized with the curvature of the mirror or lens to place the image of the slice in the appropriate depth location. This kind of display will prevent occlusion, since the virtual slices cannot block the light from those behind it. But if a single-perspective is acceptable, such as in HMDs or single-user desktop displays, occlusion can be simulated by subtracting a depth layer from those behind it (from the perspective of the observer).

While the surface of the lens or mirror is not strictly planar, slight changes in their focal length lead to large variations in the virtual image's location [DM03]. Coupled with the window-like viewing mode they enable, this means that adaptive optics-based displays can be considered flat displays.

Holography, on the other hand, works by storing the shape of the wavefront of the light emanating from the scene, by recording the interference pattern of its interaction with a clean, coherent light source. The original wavefront can then be reconstructed by illuminating the pattern with a copy of the reference coherent beam. All optical effects such as shadows, reflections and occlusions are present in the resulting image.[10]

Unlike most autostereoscopic screens, virtual volume displays can provide all the physiological depth cues (particularly accommodation), as well as continuous motion parallax. The recent advances in anisotropic screens have shortened this gap, but further properties such as vertical parallax are yet unreported in such displays, which positions virtual volume displays favorably in the realism of the 3D effect and the compactness and portability of the display system.

Deep 3D Displays

Deep displays physically occupy a volume of space and display the object inside it. Two methods can be used to implement such a system: **volumetric displays**, which place the virtual points of the object in physical 3D space, and **multi-directional screens**, which, as the name says, have either a single rotating screen, or multiple static screens facing different directions – in either case, users in a given position will see only the appropriate perspective.

Volumetric displays are omniscopic, since having the object displayed in actual 3D space allows virtually any viewpoint to get the correct perspective. Multi-directional screens will have to subdivide the perspectives into a finite number of views, and are therefore part of the multiscopic meta-category. Both can potentially implement a 360° viewing angle.

4.2.1 Volumetric Displays

Volumetric displays use several techniques to display an image in real 3D space. This means that each point of the image is actually located at the position they seem to be. This can be achieved by two main methods: static volume displays, and swept-volume displays.

Static volume displays use a substrate (solid, liquid, or gas) that is transparent in its resting state, but becomes luminous, or opaque, when excited with some form of energy. If specific points can be selectively addressed inside a volume of space filled with such a material, the activation of these points (called volumetric pixels, or voxels) forms a virtual image within the limits of the display.

Naturally, gaseous substrates are preferred, and displays have been made using artificial haze to produce unobtrusive, homogeneous clouds suspended in the air

[10] Holograms store the entirety of the information from a scene – hence their name, which derives from the Greek "holo", the same root that the word "whole" came from.

that make light beams visible. Purely air-based displays have also been proposed, using infrared laser light to produce excited plasma from the gases in the air, at the focal points of the laser. Advanced forms of such displays are common in science fiction, often mistakenly referred to as "holograms" [Hal97]. However, the actual visual quality of such displays is very far from their imagined counterparts, and even quite low compared to other current methods of 3D vision.

Swept-volume displays use a two-dimensional surface that cyclically sweeps through a volume (either moving from one extremity to another, or rotating around an axis) and display, at each point of this path, the corresponding slice of the virtual object. Due to the temporal persistence of vision, this results in what resembles a 3D object.

The main problem with volumetric displays is that, since most of the substrates used become bright when excited, rather than opaque, each point of the virtual object won't block light from the other points [Fav05], which undermines the very basic depth cue of occlusion; that is, observers would see the back side of objects as well as their front side. This is the same problem that plagues varifocal mirror displays. Such devices are therefore better-suited to display hollow or naturally semi-transparent objects, or non-photorealistic scenes – for example, icons, or wireframe 3D models [Hal97].

This difficulty could be surpassed in static-volume displays, if the substrate can be made opaque; however, a solid, static substrate would make direct manipulation and interaction with the object impossible (which is also true of swept-volume displays). The ideal volumetric display would thus be a "dynamic sculpture" that is able to change its shape and appearance according to the desired properties of the object being visualized. This has already been proposed, in a concept called "claytronics" [GCM05], but remains a strictly theoretical possibility, with no practical implementations produced so far.

4.2.2 Multi-Directional Displays

Recently, some claims have been made in the literature that the lack of occlusion in volumetric displays is not an intrinsic characteristic of the category, but a technical limitation that can be addressed.

While, as described above, this is true of static-volume volumetric displays, swept-volume displays are strictly unable to overcome this property because they work through persistence of vision, and therefore even if the active elements could be made opaque, no part of the image is permanently located in its physical position, so light would still pass through that space in the fractions of time where the display surface isn't sweeping through that particular location.

Still, swept-volume displays purported as "occlusion-capable" have been presented in recent research (for instance, [CNH+07]). They work by employing highly anisotropic diffusers to ensure that light produced or projected in the display surface is only emitted in roughly the direction the display is facing, thus ensuring that only the correct view is observed in each direction. By correctly varying the image presented in the screen according to the direction it is facing, a 3D image is produced, which can also appear to float outside the display volume.

This kind of display, however, while very similar to swept-volume *volumetric* displays, is not volumetric itself, since the image points are not located in the actual position they appear to be; in other words, they manifest the property we earlier associated with multiscopic displays, that light from each point disperses from the screen itself rather than from the correct location of the virtual point, which disables the provision of the accommodation depth cue.

These **rotating screen displays** are fundamentally similar to an earlier technique known as **cylindrical hologram** [FBS86], in which a series of images taken of a subject, with a camera performing a 360° orbit around it, are recorded in thin vertical holographic strips, which are then assembled in a cylindrical shape to provide full panoramic view of the 3D object.

In both cases, the viewer-depended variation is implemented explicitly through segmenting the viewing field, rather than producing the appropriate wavefront of the 3D scene. Cylindrical holograms, however, can potentially implement accommodation if the strips aren't holograms of a flat photograph, but of the actual 3D object itself.

5. CONCLUSIONS AND FUTURE WORK

3D displays are increasingly popular choices to provide new, more immersive and intuitive tools for education, entertainment (especially in gaming, television and cinema), telepresence, advertising, among others.

Moreover, as the technology advances, more demanding uses of such displays have started becoming feasible or expectable in the near future. Such uses require high-fidelity 3D reproductions of objects, and include areas as diverse as product design, medical imaging and telemedicine, 3D cartography, scientific

visualization, industrial prototyping, remote resource exploration, professional training and architecture.

Such wide appeal has led to the rapid development of many techniques for 3D visualization, and sometimes this has resulted in poorly-defined boundaries between techniques – especially hybrid ones. This work presented a comprehensive taxonomy of 3D displays, focusing on fundamental characteristics rather than implementation details. This property should make the taxonomy robust and expansible to include new techniques and innovations. It also provides a high-level overview of the 3D displays landscape, a useful tool for researchers entering the field.

An important property of the proposed taxonomy is that it equips both researchers and practitioners with a well-defined field map which enables application-based exploration of the 3D display space. Logically separated groups of technologies allow a faster analysis of desired properties, such as the ability to perform **direct manipulation** on the virtual objects at their apparent locations, or to **overlay** the images onto the real world, to provide augmented reality, or to operate without **headgear**. Proper **accommodation** might be crucial for high-precision applications, while support for **multiple users** is relevant in design contexts.

Furthermore, a well-defined taxonomy should also enable informed speculation over the 3D display space henceforth outlined, regarding possible new techniques and analysis of their feasibility and properties, or alternatively, discarding of a specific combination of properties (or set thereof) due to economic, physical or technological limitations. This is expected to enable new 3D display systems to be conceived. As an example, one could easily conceive a static volumetric display that provides occlusion, by using a substrate that becomes opaque when excited. This could be a relevant research topic in materials science.

This study now calls for further developments in the form of an exhaustive listing of implementations and their calatoguing in a table or database that will allow manual or automatic filtering and comparison of different display technologies and respective features.

6. REFERENCES

[Ben00] S.A. Benton. *Selected papers on three-dimensional displays*. SPIE, 2000.

[BS00] B. Blundell and A. Schwarz. *Volumetric Three-Dimensional Display Systems*. Wiley, 2000.

[CNH+07] O.S. Cossairt, J. Napoli, S.L. Hill, R.K. Dorval, and G.E. Favalora. Occlusion-capable volumetric 3D display. *Applied Optics*, 46(8):1244–1250, 2007.

[DM03] S. Dudnikov and Y. Melnikov. Review of technologies for 3D acquisition and display. Technical report, EU Project IST-2001-38862 TDIS, 2003.

[Dod05] N.A. Dodgson. Autostereoscopic 3D displays. *Computer*, 38(8):31–36, 2005.

[EucBC] Euclid. *Optics*. 300 B.C.

[Fav05] G.E. Favalora. Volumetric 3D displays and application infrastructure. *Computer*, 38(8):37–44, 2005.

[FBS86] D.S. Falk, D.R. Brill, and D.G. Stork. *Seeing the Light*. chapter 14: Holography, pages 389–391. Wiley, 1986.

[GCM05] S.C. Goldstein, J.D. Campbell, and T.C. Mowry. Programmable matter. *Computer*, 38(6):99–101, 2005.

[GW07] T. Grossman and D. Wigdor. A taxonomy of 3D on the tabletop. In *IEEE Int'l Workshop on Horiz. Interactive Hum.-Comp. Systems*, pages 137–144, 2007.

[Hal97] M. Halle. Autostereoscopic displays and computer graphics. *ACM SIGGRAPH Computer Graphics*, 31(2):58–62, 1997.

[MK94] P. Milgram and F. Kishino. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, 77(12):1321–1329, 1994.

[MZ92] M. McKenna and D. Zeltzer. Three dimensional visual display systems for virtual environments. *Presence*, 1(4):421–458, 1992.

[Oko76] T. Okoshi. *Three-Dimensional Imaging Techniques*. Academic Press, 1976.

[Tak06] Y. Takaki. High-density directional display for generating natural three-dimensional images. *Proceedings of the IEEE*, 94(3):654–663, 2006.

[UCES11] H. Urey, K. V. Chellappan, E. Erden, and P. Surman. State of the art in stereoscopic and autostereoscopic displays. *Proceedings of the IEEE*, 99(4):540–555, 2011.

[Wid01] T. Widjanarko. Brief survey on three-dimensional displays. *Media*, pages 1–27, 2001.

Approximating the Fire Flicker Effect Using Local Dynamic Radiance Maps

Jonathan Brian Metzgar
University of Colorado at Colorado Springs
jonathan@metzgar-research.com

Sudhanshu Kumar Semwal
University of Colorado at Colorado Springs
ssemwal@uccs.edu

ABSTRACT

Realistic fire and the flicker effect is a complicated process to simulate in realtime and little work has been done to simulate this complicated illumination effect in realtime. Fire is not a directionally uniform source of light but varies in intensity not only with time but also with direction. Most realtime applications use a standard point light source model for local illumination effects and may use a model to change the light source intensity with time but not direction. The problem is that point light sources are isotropic, but many sources of light have anisotropic qualities as well. Radiance maps and Precomputed Radiance Transfer (PRT) have been used to increase realism at realtime interactive frame rates. These models approximate global illumination by applying an environment map (typically approximated with spherical harmonics) to get their soft lighting effect. In this paper we present Local Dynamic Radiance Maps (LDRM) which uses radiance maps in a local illumination model to add anisotropic behavior to light sources. We implemented a realtime rendering engine that supports shadow mapping and the physically based Cook-Torrance model to approximate global illumination. In particular, we generate dynamic radiance maps using Perlin noise to simulate the nonlinear radiance of fire and we also implement a rudimentary Lattice-Boltzmann flame rendering effect. Finally, we show how LDRM can be applied not just to approximating the fire flicker effect, but as a general framework for simulating the illumination properties of other nonlinear light sources.

Keywords: radiosity, global illumination, fire, Lattice-Boltzmann, radiance maps, shadow-mapping.

1 INTRODUCTION

Advances in graphics processing units (GPU) have resulted in not only improved speed and quality of computer generated images, but now feature massively parallel processors capable of running several general purpose programs. This parallelism allows for the implementation of global illumination algorithms. Physical simulations of natural phenomena like fire and water are taking advantages of the hardware acceleration.

Rendering fire is a big challenge for computer graphics because it touches so many areas of image generation. It is even harder to do it well in realtime. One would want to eventually render fire based on a full 3D simulation using Navier-Stokes equations and render a scene in realtime using the illumination effects modeled by a radiosity algorithm. Since this is not practical, fire imagery is often created through precomputed renderings, video, or particle effects and the illumination comes from a point light source with a dynamic intensity. But, simply modulating the intensity and location of a point light source does not adequately model the way that fire radiates in a nonlinear way. It is this nonlinear radiance that causes the flicker effect to occur.

For the last decade, radiance maps and precomputed radiance transfer have become essential for creating high quality realtime visualizations. Essentially by utilizing approximations like spherical harmonics, they can quickly apply the illumination model to objects in a scene and get radiosity like shading effects. This model works with an outward-in approach where the incoming light is mapped to a sphere which is mapped to the precomputed radiance map. We propose a variation of this method that 1) dynamically computes the radiance and 2) is a local source of radiance which can move around inside an environment. We call these local dynamic radiance maps or LDRM. The main goal of the LDRM model is to make lights appear and act like they belong in the scene by modeling their anisotropic behavior as a function of time.

In this paper, we present a new way to approximate the flicker effect by procedurally generating a LDRM into a cube map. This LDRM is projected outwards from the position of the fire source through the cube texture and onto the surrounding scene's geometry simulating

the first bounce of radiosity. To achieve our results, we have chosen bump mapping, physically based lighting, and soft shadow mapping algorithms to calculate direct and indirect illumination of the fire source. Finally, we implement a procedurally simulated fire effect based on a Lattice-Boltzmann model which can be simulated on the GPU or CPU to approximate the flames emitted from a torch. This is rendered in our scene at the location of our fire source.

2 PREVIOUS WORK

Simulating fire is a fluid simulation problem that has a variety of solutions ranging from artist derived flame profiles, procedural generation of flames, and fluid simulation of flames. [Ngu01a] and [Ngu02a] present a solution for Navier-Stokes equations to simulate the flames but is time consuming. [Hon07a] combine both a Navier-Stokes simulation with detonation shock dynamics to get wrinkled flames and cellular patterns but also takes a long time to render. A near realtime fire simulation and control system that uses artist derived flame profiles is described by Lamorlette and Foster in [Lam02a]. The Perlin noise function and an artist specified flame profile is used to generate procedural fire by [Ful07a].

The groundbreaking 1970's work by [Har73a] and [Har76a] first introduce the Lattice-Boltzmann Model (LBM) methods which have become the basis of many non Navier-Stokes fluid simulations. [Wei02a] use overlapping textures with turbulence details employing LBM for motion of the flames. [Zha03a] also employ LBM to simulate fire fronts around solid objects.

Some models, such as [Lam02a], discuss in much detail how to compute the lighting effects. For example [Lam02a] uses an emitting sphere at each flame segment to generate lighting. It is assumed that the lighting details are automatically handled at the renderer level which combine several global illumination algorithms like radiosity, ray tracing, and/or photon mapping. Importance sampling using volumetric illumination has been used by [Zha11a]. GPU simulation with volumetric data creates a variety of realistic and detailed fire simulation such as moving fire [Hor09a],

Radiance maps are images that store the intensities of light passing through each pixel. The direction of the light is determined by the projection used to create the image. They are used in high dynamic range imagery (HDRI) as described by [Deb97a]. Methods such as precomputed radiance transfer (PRT) first introduced by [Slo02a] use a spherical harmonics representation for low-frequency radiance computation. These spherical harmonics representations approximate a high resolution HDRI radiance map and are very effective for relighting objects in an environment. Primarily they have been used for static scenes but the technique is expanding for dynamic scenes as well. For example, [Kri05a] relight architectural models in real-time with moving lights by combining precomputed point light source clouds.

3 THE POINT LIGHT MODEL

The predominantly implemented illumination model for fire in realtime applications is the point light source model. It is used widely because there is hardware support for the algorithm and also because it is easy to compute the shading value since the light position is subtracted from the vertex or fragment position to get the $\vec{L}$ vector that is used by a Lambertian illumination model. In some implementations, the intensity is constant with a distance based falloff function. It can become more sophisticated by varying the intensity of the light or adding a random perturbation to the coordinates of the light source. The intensity and position are often varied using a smooth noise or interpolation scheme. The easiest way to think of this is a person holding a simple light bulb with a rapidly sliding dimmer switch and a jittery hand. In Figure 1, you can see the smooth uniform intensity that the point light model has. The problem is that point light sources are isotropic, but many sources of light are anisotropic. We will now present the LDRM model which attempts to model the anisotropic features that fire and other natural phenomena possess.

4 THE LDRM MODEL

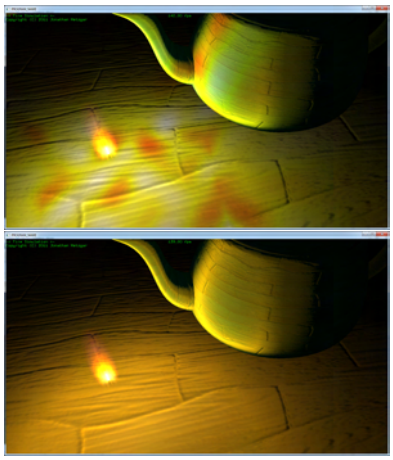


Figure 1: The LDRM method is compared to the point light source method. In the LDRM image, the intensity and frequency is turned up very high to clearly show the difference of the LDRM method and the point light source method. Realistic settings would be tuned to be more subtle.

Our Local Dynamic Radiance Map (LDRM) model stores a dynamically computed radiance map at coordinates P. The light emitted from P is cast in all directions simulating the first bounce of radiosity. The radiance may either be precomputed as an animation

or procedurally generated in realtime. A cube map or other similar abstraction is an ideal way of storing these radiance values.

We reproduce Kajiya's rendering equation [Kaj86a] below so we can illustrate how the LDRM fits into this standard model:

$$I(x,x') = g(x,x')[\varepsilon(x,x') + \oint \rho(x,x',x'')I(x',x'')dx'']. \quad (1)$$

The radiance of the LDRM is represented by $\varepsilon(x,x')$ while being directly affected by the visibility of the point x at point x' by the function $g(x,x')$. More specifically, the function $\varepsilon(x,x')$ is the radiance coming from direction $\vec{L}$ where $\vec{L}$ is the vector from the position of the light source to the point x, and is the direction of the sample. This value can be obtained by looking into the radiance map using the direction provided. For example, most graphics hardware have the ability to easily look up this value from a cube map using a vector as an input.

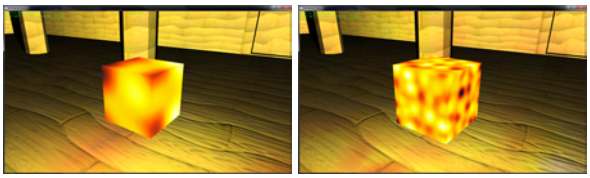


Figure 2: Two LDRMs generated using different frequencies of noise and their corresponding effect on the environment.

From Kajiya's rendering equation, we then map this to a simplified model where we can incorporate our rendering algorithms. Since we are not simulating the integral in his equation, we decided to make that a constant value and focus on just ε and g which is just the radiance of the light source and the visibility of the light source with the surface being illuminated. Essentially, the LDRM acts like a Gaussian surface in the sense, that instead of trying to compute the interactions with the actual flames or other phenomena and the surrounding environment, we perform an intermediate step of mapping it to a surface we can easily use in a rendering situation. This is clearly seen in Figure 3.

The incoming radiance which we will now call R, is then divided into the specular and diffuse reflection colors $k_{specular}R$ and $k_{diffuse}R$, respectively where $k_{specular} + k_{diffuse} = 1$. Depending on the reflectance model used, $k_{specular}$ and $k_{diffuse}$ may be computed differently. We decided to implement the Cook-Torrance model and we will discuss later in section 4.1 how to compute these values.

The dynamic radiance of a torch fire is approximated by using Ken Perlin's noise function. The radiance of each direction of the torch fire is computed and stored inside the radiance map. The unit vector l is used as input to the Perlin noise function and the resulting value is used to look up the color associated with the radiance.

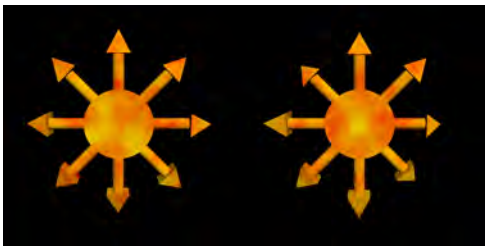


Figure 3: The LDRM method (right) differs from the point light source method (left) by modeling the light's outgoing radiance as a function of time, intensity, *and* direction.

A blackbody radiation color map is used to give color to the torch fire. An example color map is shown in Figure 7.

The LDRM is flexible. If a variety of natural phenomena used the same kind of simulation algorithm but differed only in color, a different color map will easily allow for adjusting that. For example, fire could probably use the same simulation code, but the specific chemical combustion properties would be approximated by mapping the resulting intensity values with the appropriate color map.

In our implementation, we have chosen Perlin noise because it generates smooth noise that can be animated and provides enough variety for our ideas to be implemented. Generation of a LDRM function that simulates the unique properties of fire (or other phenomena) is most definitely a topic for future study but is outside the scope of our research. Later we will discuss this possibility, but our experiments with Perlin noise showed significant enough improvement in scene realism versus the traditional point light source method that we discussed in the last section.

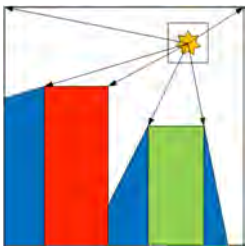


Figure 4: The LDRM projects radially from the center of the light position. Areas in blue get ambient lighting while others get direct illumination.

Figure 4 shows a diagram of how the LDRM method works. The box located around the position of the light represents the cube map. The arrows emitted from the center of the light through the box will look up the appropriate radiance and project it on the environment. If the area is in shadow (represented by shaded blue areas) then an ambient algorithm can determine the final illumination of those fragments. The containing rectangle

and the red and green rectangles represent the environment and objects visible to the light source. Figure 2 shows two example LDRMs generated using two different frequencies of noise. It can be easily seen how the LDRM works in a practical sense by observing that variation in intensity in the environment corresponds to the frequency of noise.

5 IMPLEMENTATION

Figure 5: A rendering of the fire flicker effect program.

Our fire flicker effect simulation presented in this paper is designed to employ the LDRM model. A variety of rendering algorithms is used to simulate global illumination and a screenshot is shown in Figure 5. The global illumination algorithm implements the Cook-Torrance model, Blinn bump mapping, and cube map shadow mapping. It uses a simple ambient function that approximates indirect illumination by scaling the direct illumination by the amount of shadow present at that pixel location. The Cook-Torrance model allows for physically based illumination while the bump mapping algorithm allows for increased higher-frequency pseudo details. Finally instead of rendering the LDRM cube map in the scene, flames are dynamically computed and rendered into 2D textures and drawn onto rectangles in a fan like structure to give the torch a 3D look. In effect, the torch fire is used as an aesthetic place mat to show where the LDRM is located in the scene.

5.1 Illumination and Shadow Model

The Cook-Torrance model was chosen because it is a physically based model. Other models can easily be integrated as desired. The LDRM was used to supply the specular color $k_{specular}R$ for the Cook-Torrance model. This color is mixed in with the surface color of the fragment being rendered and scaled by the dot product of the surface normal and incoming light vector $\vec{L}$.

Since a torch is an omni-directional light source, cube mapped shadow mapping was selected to render the shadows. It is a fairly straightforward algorithm to implement, but it does take some tweaking to get the highest image quality. We chose to write a scalable multi-sampled shadow algorithm that we could adjust to measure performance of our technique. The number of samples can go from one sample to $N = 257$ samples. Here N can be varied based on the hardware capabilities of the system. The penumbra width is adjustable as a constant parameter in the shader program. Figure 6 shows two screenshots of the program using 1 sample shadows and 257 sample, wide penumbra shadows.

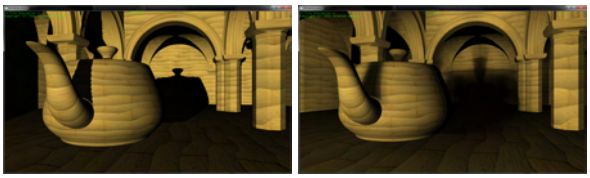


Figure 6: These two images shows a basic one sample shadow and a wide penumbra, 257 sample shadow.

5.2 Radiance Cube Map Generation

Perlin noise is a simple solution to generating non-linear radiance that is repeatable, smooth, and fluid. Depending on application performance, the noise can either be generated on the GPU or CPU. The fire program implemented in this paper used the GPU and a render-to-texture set up to render the six sides of a cube map. The GPU code for generating Perlin noise was implemented by [Gus06a] which we slightly modified to adjust for noise scaling and animation parameters used in the fire program.

The six textures are used as a cube map in the final rendering pass by the global illumination shader. The gray scale output of the radiance is then converted from heat values to RGB values by looking up the data in a color look up table. The texture is updated once per frame or as needed to maintain a target frame-rate. The color look up table is shown in Figure 7 and one side of a LDRM is shown in figure 8.

Figure 7: The color look up table mapping heat to their corresponding RGB values.

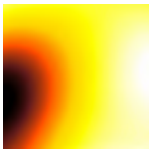


Figure 8: One face of a cube map generated by the LDRM method.

Special care needs to be taken to balance the noise so that it adds a subtle lighting effect to the scene. If the frequency of the noise is too high then the effect may look overdone where it can quickly be distracting. On the other hand, using hardly any noise or under using the effect will look as if the effect is not being used, so careful balancing needs to done to find a good range where the effect will be effective. Figure 2 shows this effect in

practice. Note how the high frequency map may make the environment look splotchy which is not realistic.

The LDRM may be used to create good lighting effects, but it is not complete without some motion of the shadows in the environment. Perlin noise is used once again to compute a time-varying offset which we add to the original position of the light. This new position is used to render the shadow maps and lighting. The final product then has dancing shadows which enhance realism.

5.3 Global Illumination in the Simulation

The equation

$$C_{out} = \max\left(R_{ambient}, R_{shadow}\right) \cdot \left(k_{diffuse} + R_{specular} \cdot k_{specular}\right). \quad (2)$$

is the basis for the global illumination algorithm for the fire program. The $R_{ambient}$ term specifies the ambient intensity of the pixel, the R_{shadow} represents the contribution of any direct lighting occuring at the pixel, the $k_{diffuse}$ term is the color of the surface at that pixel, the $k_{specular}$ term is the color of the specular reflection of the pixel, and the $R_{specular}$ term is the amount of reflected light at the pixel.

The terms are all computed from four different algorithms. The first algorithm is bump mapping which calculates the normal of the pixel. The second algorithm is the Cook-Torrance model which calculates the specular reflectance values $R_{specular}$ and $k_{specular}$ of the pixel. The third algorithm is the Cube Map Shadow algorithm which allows for omni-directional point light sources. Finally, the fourth algorithm is an intensity falloffmodel for the $R_{ambient}$ term to model indirect light. These have been covered in detail in previous works, but integrating them together will be briefly explained in light of the equation to compute C_{out}.

5.4 Ambient and Shadow Term

The ambient term is a simple approximation based on a inverse falloff law from the distance to the fire. The ambient term R_a is computed by the formula

$$R_{ambient} = \frac{1}{4|L|}. \quad (3)$$

This equation is a variation based on the inverse power law $I = \frac{P}{4\pi r^2}$ which gives us a brighter overall light intensity which is normally lost unless you do a full on radiosity simulation to get the intensity back through indirect reflections. This gives us some of that light which is normally "lost" in a local illumination model.

The shadow term R_{shadow} is computed with the following formula:

$$R_{shadow} = \min\left(\vec{N}_{vertex} \cdot \vec{L}, \vec{N}_{bump} \cdot \vec{L}\right) * \frac{1}{n} \sum_{j=0}^{n} s_j \quad (4)$$

where $\vec{N}_{vertex}$ is the interpolated vertex normal, $\vec{N}_{bump}$ is the per pixel normal derived from the normal map, $\vec{L}$ is the incoming direction of the light source, n is the number of samples being used for the shadows, and s_j is the boolean result of comparing the jth pixel depth value to the light depth buffer which is either 1 or 0. Taking the minimum of the dot products eliminates bump mapping on polygons not facing the light.

Together the ambient and shadow terms are used to determine the minimum illumination level of the fragment to be rendered. A simple maximum function is used to choose the ambient term or shadow term. If a fragment is completely shadowed, then the ambient term is used, otherwise the fragment is in penumbra and has some illumination.

5.5 Diffuse and Specular Term

The diffuse term $k_{diffuse}$ is generated from the surface color or texture of the object. The specular terms $R_{specular}$ and $k_{specular}$ are the coefficient of the reflected light and its color, respectively. This is where we can incorporate the LDRM model. The $k_{specular}$ value is obtained by using the $\vec{L}$ vector as the lookup in the LDRM cube map. The $R_{specular}$ term is based offthe Cook-Torrance model equation [Coo81a]

$$R_{specular} = \frac{F}{\pi} \frac{DG}{(N \cdot L)(N \cdot V)}. \quad (5)$$

F is the Fresnel term, D is the micro-facet distribution factor, G is the geometric attenuation factor, V is the view vector, and L is the vector from the light to the fragment. Additional details about using the Cook-Torrance may be found by referring to the original paper. It is important to note that the LDRM model is not just limited to Cook-Torrance, but may be incorporated with any illumination model.

5.6 CPU and GPU 2D Flame Simulation

Our flame rendering system is based off a simple cellular automata model to generate fire. This cellular automata is a simplified model of Lattice Boltzmann Methods (LBM) which originated with the work of [Har73a]. [Che98a]'s work summarize the developments of the model into its more current form. The method works by using a lattice structure representing the fluid to be simulated. A *convection operator and* collision operator transform the lattice over time and cause the fluid process to occur. The nineties demo scene fire effect used a simple averaging function to cause convection and simulate collisions. Figure 9 shows a screenshot that simulates this full screen fire effect.

This fire effect can be modified to generate small flames or torch sources. Further improvements can be made to increase precision as well. Typically this effect uses 8-bit integer mathematics to store the heat values. While

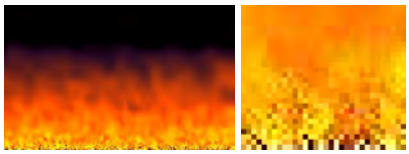


Figure 9: This image shows a fire simulation where the entire bottom row is used as the heat source and that use of integer math causes noisy artifcats near these randomized heat sources.

this is accurate enough for most of the effect, it results in artifacts near the source of the fire as shown in Figure 9. Changing the representation from integers to floating point math eliminates these artifacts and increases dynamic range. This can be coupled with established HDR techniques and physically based color computations for different chemical reactions.

The flame is generated by adding or seeding heat to points on the lattice. The flame will flow during the convection operator step. During the collision operator step, the flame mixes together. The three steps will cause the flame to take shape as this process repeats. A simple circular falloff model is used for seeding the heat to the fire. Notice in Figure 11 two different kinds of falloff patterns: the simple radial falloff used in the fire program and a noisy radial falloff used for the fires in Figure 10. By quickly changing the location of the falloff pattern, turbulence is created. The fire effect and varying levels of turbulence are shown in Figure 10.

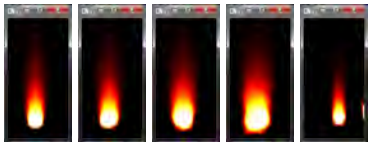


Figure 10: The radius of the circle in which the center of the flame source is moved causes a more turbulent flame. On the far right, improperly handled edges cause “heat sink” artifacts.

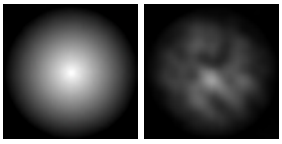


Figure 11: Two falloff patterns for seeding fires. The second pattern adds some turbulence to the resulting flames.

This fire effect can be computed using a GPU and a graphics based shader language (i.e. GLSL) was adequate for our simulation. The algorithm is shown in Listing 1 and is fairly straightforward. It determines whether the fragment is in the simulation area or not (which if not handled correctly creates “heat sinks" shown on the far right in Figure 10). Simulating diffusion and cooling is obtained by averaging several neighbor samples at each fragment and multiplying by a factor $life$, respectively. Heat is added to all the fragments located inside a circle (a “heat sink") which is randomly jittered according to the desired turbulence of the flame.

```
@VERTEXSHADER
uniform mat4 ProjectionMatrix;
varying vec2 uv;

void main() {\\
uv = gl_MultiTexCoord0.st;
    gl_Position = ftransform();
}

@FRAGMENTSHADER
#version 140
uniform sampler2DRect FireLattice;
uniform sampler1D radianceCLUT;
uniform float a, b, radius;
uniform float width, height;
uniform float heat, life;
uniform float turbulence;
in vec2 uv;
out vec4 gl_FragColor;
float rand(vec2 co) {
  return fract(sin(dot(co.xy ,vec2(
    12.9898,78.233))) * 43758.5453);
}
void main() {
  float x = uv.s, y = uv.t;
  float data = 0;
  float r2 = radius * radius;

  if (x >= 1 && x < width-2 &&
      y >= 3 && y < height-1) {
    if (x >= a-radius && x < a+radius &&
        y >= b-radius && y < b+radius) {
      float f = (x-a)*(x-a) + (y-b)*(y-b);
      if (f < r2) {
        data = texture(FireLattice,
                       vec2(x, y)).a;
        data += heat * (1 - f/r2);
     }
    }
    data+=texture(FireLattice,
                  vec2(x, y+1)).a;
    data+=texture(FireLattice,
                  vec2(x-1, y-1)).a;
    data+=texture(FireLattice,
                  vec2(x+1, y-1)).a;
    data+=texture(FireLattice,
                  vec2(x, y-2)).a;
    data = clamp(data * life / 4.0,
                 0.0, 1.0);
  } else {
    data = 0;
  }
  vec3 color2 = texture(radianceCLUT,
                        data).rgb;
  gl_FragColor = vec4(color2,data);
}
```

Listing 1: A GLSL Shader that computes the flame simulation.

6 RESULTS

The LDRM model takes up relatively little extra load in conjunction with normal rendering depending on the number of lights being used. The majority of performance loss comes from shadow mapping when large numbers of samples are being used. Rendering high resolution LDRM cube maps may also reduce performance but this can be mitigated by using low resolution maps when large numbers of lights are being used.

The benchmarks were conducted using a Windows 7 OS, Intel i7 930 2.80GHz processor with 6GB of RAM, and a NVIDIA GeForce GTX 480 graphics card with 1.5 GB of GDDR5 memory. Each benchmark was measured by recording the number of frames per second (FPS) once per second over a period of 25 seconds. The mean frame rate was then computed to filter noise in the readings, though the noise present was so low that it had an insignificant effect on the final numbers. Finally, we kept the frame rate as high as possible so we could ensure that our simulation would run on less capable graphics cards.

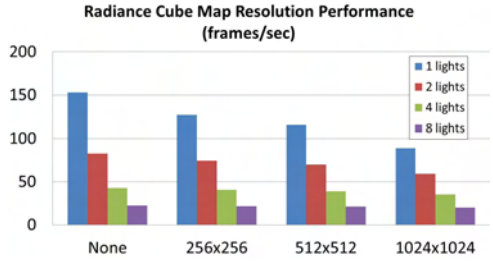


Figure 12: A comparison of radiance cube map size versus performance.

We tested the performance of our method using different resolution LDRM cube maps and by not rendering them at all. Figure 12 shows the results when 1, 2, 4, or 8 lights are being used. When using a 256x256 LDRM cube map, performance drops by 41%, 68%, and 83% for 2, 4, or 8 lights, respectively. When using 512x512 cube maps, performance drops by 39%, 66%, and 82% for 2, 4, or 8 lights, respectively. For 1024x1024 cube maps, performance drops by 33%, 60%, and 77% for 2, 4, or 8 lights, respectively. Compared to not using LDRMs at all, performance drops by 4% to 17% for 256x256 cube maps, 6% to 25% for 512x512 cube maps, and 11% to 42% for 1024x1024 cube maps.

Next, we tested the performance of our flame rendering system on the CPU and the GPU. Overall, the GPU had a clear lead in performance especially as resolution is increased. However, until much higher resolutions of flame simulations are used, the number of flames rendered per second on the CPU was in the hundreds which is sufficient. It is also clearly shown that using the GPU in conjunction with the CPU yielded little decrease in overall performance. We are unable to easily compare

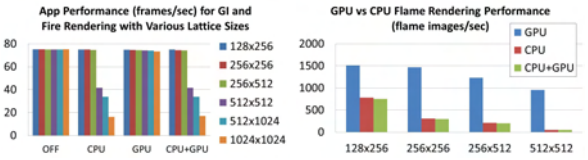


Figure 13: On the left, we see that application performance is not affected until high resolution lattice sizes are used. On the right, the GPU far surpasses the CPU in raw fire rendering performance.

our flame rendering algorithm with others because ours is not volumetric and has very strong boundary conditions which make it incapable of handling interactions in a 3D environment which a volumetric simulation could.

When integrated with the global illumination simulation, the GPU advantage becomes more obvious. CPU performance drops off fast when using high resolution lattice simulations, though it is almost unnoticeable when using reasonably sized maps. In contrast, the GPU simulations have a very small performance penalty when using large lattices. It should be noted that multithreading was not used in the CPU simulation, but the GPU still has enough compute power for the large lattice sizes that even an 8 core CPU could not outperform it. Figure 13 shows the performance graphs for running the global illumination simulation and rendering the flames with either the CPU, GPU, or both. It also shows the baseline performance of the GI simulation without rendering the flames. Effectively, you get the flame rendering for free for small resolution flame images.

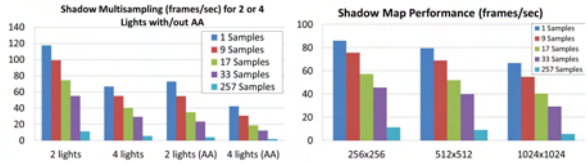


Figure 14: On the left, antialiasing halves overall performance. On the right, reasonable numbers of shadow samples still allow interactive frame rates.

Finally, we examine the performance of the global illumination algorithm. Figure 14 shows the performance of the global illumination algorithm both when using multi-sampled shadows and different resolution shadow maps, respectively. When using a reasonable number of lights, it is very easy to obtain very interactive rates. Performance drops quite a bit when using anti-aliasing, but the image quality is greatly improved and small pixel artifacts that show up when not using anti-aliasing almost entirely disappear.

Shadow quality is very good at 33 samples per pixel and the frame-rate is quite interactive. Wide penumbras are allowed which increases the realism of far off shadows where the area lighting effect of the flames would not

create sharp edges. The shadow map size affects performance but not as dramatic as varying the number of samples. Memory usage does increases quickly so tweaking is necessary to determine the lowest acceptable shadow map resolution.

Adding motion to the shadows does a good job of distracting the observer from noticing some minor problems with shadow mapping. Some of these problems include light leakage or surface acne. Ultimately, the moving shadows create the realistic appearance that the fire has on the scene while the LDRM model adds a subtle ambience to the scene, that when switched off, makes the simple intensity modulation based fire flicker effect seem somewhat lifeless.

7 CONCLUSION AND FUTURE WORK

The LDRM model presented in this paper helps add realism to scenes where torch fires are being used. The ambiance created by using shifting anisotropic illumination patterns add subtle depth and realism to scenes compared to the simple point light source model. The performance penalty is small and the algorithm is trivial to implement for any realtime graphics engine.

Future study of LDRMs to enhance direct illumination is promising and is an excellent extension to normal precomputed radiance transfer. In the future we are looking into simulating a volumetric fire and comparing the actual radiance with our approximation. We believe that creating LDRM models of other nonlinear light sources would be highly beneficial towards accurately simulating other phenomena in a realtime application. Finally, we are looking into using spherical harmonics as a substitute for cube maps which may allow LDRMs to be used in resource limited environments.

8 REFERENCES

[Che98a] Chen, S., and Doolen, G.D., *Lattice boltzmann method for fluid flows.* Annual Review Fluid Mechanics, 1998, pp.329-364.

[Coo81a] Cook, R. L., and Torrance, K. E., *A reflectance model for computer graphics.* Proceedings of the 8th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '81, 1981, pp. 307–316.

[Deb97a] Debevec, P. E., and Malik, J., *Recovering high dynamic range radiance maps from photographs.* Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '97, 1997, pp. 369–378.

[Ful07a] Fuller, A. R., Krishnan, H., Mahrous, K., Hamann, B., and Joy, K. I., *Real-time procedural volumetric fire.* In Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM, New York, NY, USA, I3D '07, 2007, pp. 175–180.

[Gus06a] Gustafson, S., *Dsonoises, a set of useful functions for sl.*, 2007. url:http://staffwww.itn.liu.se/~stegu /aqsis/DSOs/DSOnoises.html

[Har73a] Hardy, J., Pomeau, Y., and de Pazzis, O., *Time evolution of a two-dimensional classical lattice system.* Phys. Rev. Lett. 31, 5, 1973, pp. 276–279.

[Har76a] Hardy, J., de Pazzis, O., and Pomeau, Y., *Molecular dynamics of a classical lattice gas: transport properties and time correlation functions.* Phys. Rev. A 13, 5 (May), 1976, pp.1949-1961.

[Hon07a] Hong, J.-M., Shinar, T., and Fedkiw, R., *Wrinkled flames and cellular patterns.* In ACM SIGGRAPH 2007 papers, 2007, ACM, New York, NY, USA, SIGGRAPH '07.

[Hor09a] Horvath C and Geiger W., *Directable high Resolution simulation of fire on the gpu.* In ACM SIGGRAPH 2009 papers, 2009, ACM, New York, NY, USA, SIGGRAPH '09, 28(3).

[Kaj86a] Kajiya, J. T., *The rendering equation.* In ACM SIGGRAPH 1986 papers, ACM, New York, NY, USA, SIGGRAPH '86, 1986, pp. 143-150.

[Kri05a] A. W., Akenine-Möller, T., and Jensen, H. W., *Precomputed local radiance transfer for real-time lighting design.* ACM Trans. Graph. 24, July 2005, pp. 1208-1215.

[Lam02a] Lamorlette, A., and Foster, N. *Structural modeling of flames for a production environment.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '02, 2002, pp. 729-735.

[Ngu01a] Nguyen, D. Q., Fedkiw, R. P., and Kang, M. *A boundary condition capturing method for incompressible flame discontinuities.* Journal of Computational Physics 172, September, 2001, pp. 71–98.

[Ngu02a] Nguyen, D. Q., Fedkiw, R., and Jensen, H. W. *Physically based modeling and animation of fire.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '02, 2002, pp. 721–728.

[Slo02a] Sloan, P.-P., Kautz, J., and Snyder, J. *Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments.* Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, SIGGRAPH '02, 2002, pp. 527–536.

[Wei02a] Wei, X., Li, W., Mueller, K., and Kaufman, A. *Simulating fire with texture splats.* Proceedings of the conference on Visualization '02, IEEE Computer Society, Washington, DC, USA, VIS '02, 2002, pp. 227–235.

[Zha03a] Zhao, Y., Wei, X., Fan, Z., Kaufman, A., and Qin, H. *Voxels on fire.* Proceedings of the 14th IEEE Visualization 2003 (VIS'03), IEEE Computer Society, Washington, DC, USA, VIS '03, 2003, pp. 36.

[Zha11a] Zhang, Y., Zhu, D., Qiu, X., Wang, Z. *Important Sampling for volumetric illumination of flames.* Visual Computing in Biology and Medicine, VR in Brazil, Computer & Graphics, 35(2), 2011, pp. 312-319.

A Matching Shader Technique for Model-Based Tracking

Martin Schumann, Jan Hoppenheit, Stefan Müller
University of Koblenz-Landau
Institute of Computational Visualistics
56070 Koblenz, Germany
{schumi, silver, stefanm}@uni-koblenz.de

ABSTRACT

We present a line feature matching method for model-based camera pose tracking. It uses the GPU for computing the best corresponding image line match to the edges of a given 3D model on a pixel basis. Further, knowledge about the model is considered to improve the matching process and to define quality criteria for match selection. Each edge is rendered several times with image offsets from the last estimated position of the model. The shader counts the number of pixels in an underlying canny-filtered camera input image. Returning the best fit by pixel count can be done applying occlusion queries. A speed-up can be achieved using a more elaborate shader with texture read-back reducing the number of rendering passes. The matching shader is not limited to work with lines and can be extended to other structures as well.

Keywords

Model-Based Camera Pose Tracking, Line Feature Matching, GPU Shader.

1 INTRODUCTION

Camera pose tracking is the process of estimating the viewing position and orientation of a camera. This can be performed using a model of the environment represented by 3D data available from a modeling process or created online. Using a model leads to more stable tracking without drift occurrence, as it is the case for frame-to-frame tracking. Further the model serves as an absolute reference for initialization.

The pose estimation problem is based upon establishing 2D-3D correspondences between features of the model and features in the camera image that may be points, lines or higher structures. The aim is to minimize the distance between projected 3D features and their 2D correspondences in the camera image. Establishing these correspondences is crucial for estimating a good camera pose. False matches lead to shifting in the pose, jittering or even loss of the tracking.

Tracking on CAD models was realized by [Com03], and respectable success in combination of edges with texture information could be demonstrated by [Vac04]. Current research is focused on SLAM (Simultaneous Localization and Mapping) algorithms [Kle07], where feature cloud maps are reconstructed from the visible surroundings.

In the approach of analysis-by-synthesis even further knowledge about the model is used for the tracking process. Beginning from an initially estimated pose or the pose of the given 3D model in the last image, a rendered image or a structure of features is synthesized together with a collection of additional information available from rendering process or from global knowledge. In the analysis step these are compared to a real camera image to estimate the current camera pose. In [Wue07] they use depth and normal information to derive the line trait of the model. The work of [Sch09] analyzes similarity-based and feature-based methods for comparing synthetic and real image and [Bra11] simulate the lightning conditions to improve tracking.

We present a method for matching model edges to lines in the camera image using the GPU. It uses the model knowledge to define the quality of the matches for match selection. In our approach we work with straight lines but the technique is not limited to this type of feature and can be used for other structures as well.

2 RELATED WORK

The problem of matching and registration of images does not only appear in camera pose tracking but also in applications of object recognition and image registration e.g. for medical purposes. In our approach we want so solve for the 3D pose of a camera in a model-based tracking system. What we are focusing on, is a method for line feature-based model-image matching so that the knowledge about the model geometry and

perspective can be used to improve the correspondences and to define quality criteria, which is not a usual task.

Possible approaches for matching are intensity-based similarity measures regarding the entire image or patches of it, analysis of the image in the frequency domain or discrete image features like points and lines, describing visually perceivable structures in the image.

Detection and matching of feature points has been developed for a long time. First, the locations of interesting points like edge crossings or corners are detected in the image. The pixel surrounding of an interest point is then described by a vector of intensities, and may also include scale and orientation as SIFT [Low99] and SURF [Bay08] do. Matching is realized by comparing the entries of these descriptors which may be very time consuming due to scale space calculation. For acceleration there exist GPU-based implementations of feature detection, matching or tracking algorithms as the well-known KLT [Shi94] by [Sin06].

Feature edges can be detected by common image processing filters like the Sobel operator or more advanced developments as the canny algorithm [Can86]. Sobel and Canny implementations using the GPU in the context of a particle filter framework are shown in [Kle06] and [Bro12]. Line matching is mainly realized by minimizing the Euclidean distances between the projected model edges and corresponding gradients in the image. A simple distance measure may be gained by projecting the start and end point of the model edge to the image line or matching in parameter space. However, this requires a parameter transformation as Hough [Dud72], which may be expensive. In [Low91] simply the perpendicular distances of the projected model and the 2D image segments are used and in [Low92] a combination of distance and orientation is proposed.

Another popular distance-based matching method is the Moving Edges algorithm [Bou89]. It is used in various tracking frameworks as shown in [Har90],[Dru02],[Com03] or [Vac04] to name some of them. The model edge is sampled for control points and from these, orthogonal search lines are spanned in both directions. Alongside these line normals the gradient maximum of the image is calculated and the distance between 3D control point and 2D image point found is minimized. To deal with possible multiple gradient maxima the approach can be improved using multiple hypotheses for each sample point which provides higher stability [Vac04][Wue05].

3 THE MATCHING SHADER

3.1 Shader Outline

The model-based tracking approach uses a 3D model of the object to be tracked. Model edges can be obtained from this model by rendering an image with the last pose, detecting lines in the image and back-projecting to the model in order to gain 3D coordinates. Instead we use the model data structure directly by selecting individual edges and performing a visibility test. So the image processing step on the rendered image can be omitted. The advantage of a candidate edge list is that the matching result can be sorted and weighted by the quality of the matches.

For these 3D model edges corresponding 2D line matches should be found in the camera image. This camera image is canny-filtered so that natural structures are expressed as a binary image. The model edges selected for matching are projected and rendered with a matching shader from the pose of the last estimation with frame buffer write disabled. For each drawn pixel of the model edge the called pixel shader reads the value of the underlying canny image at the pixel position. If there is a black edge pixel in the canny image, the shader outputs a color. Otherwise it is discarded and the render pass will interrupt. The concept is displayed in figure 1 and listing 1 shows the matching shader in GLSL code.

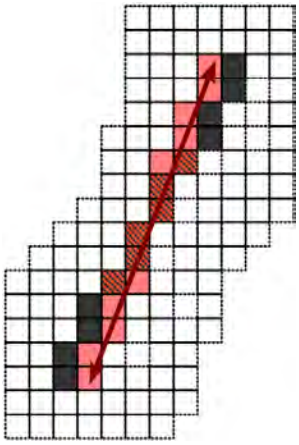


Figure 1: Rendered edge (red), image pixel edge (black) and common pixel to be counted (hatched).

The number of successful render passes now corresponds to the number of image line pixel counted. Retrieving this result number can be done by running occlusion queries while rendering (Section 3.3). The pixel count itself tells us about the probability that a found line in the canny image corresponds to a model edge. The number of counted pixel is a measure of the line length. Ideally the matching shader count equals the model edge length.

Using information of the model can help to improve the matching process. From the known pixel length of the rendered model edge we can expect a certain length of the image line response and thus define a threshold for a minimum pixel count. If the image line found does not fulfill this minimum length, it will be rejected as corre-

Vertex shader

```
void main()
{
    gl_Position = ftransform();
    gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

Fragment shader

```
uniform sampler2D cannyImg;
vec3 val;
void main()
{
  val = texture2D(cannyImg, gl_TexCoord[0].st).xyz;
  if(( val != vec3(1.0,1.0,1.0) ))
      gl_FragColor = vec4(0.0,0.0,0.0,1.0);
  else
      discard;
}
```

Listing 1: Counting shader.

spondence. In the next section we show further criteria for evaluation of the matches like depth and distance.

3.2 Sample Edge Generation

While we assume small movements of the camera from one image to the next, the model edges must be varied in position and orientation covering translations and rotations of the camera. This is done by sampling several new image edges around the known projected model edge. For each model edge start and end point in the image are known. Around these new points are sampled with an offset, e.g. on a 3x3 window around the central model edge start and end point, 8 new possible points are generated for each one. All generated points, including the original ones, are then connected to new edges, resulting in a total of 81 candidate edges in this case. Figure 2 and 3 show an example for some sample edges covering possible translations and rotations of the model edge. Notice that the sampling is done in 2D image space for projected edges. The 3D model data itself remains rigid.

For all of these candidate sample edges the matching shader returns a number of pixel counted as described in Section 3.1. The candidate edge returning the highest pixel count can be regarded as the best fitting match. Beneath the expected pixel length, it is even possible to consider the distance between the pixel count results of each candidate as quality criterion for matching. Similar parallel lines will return almost equal numbers of pixel count and thus can be recognized as ambiguous features. Such results may be rejected for matching.

Choosing the offset depends on accuracy and computational speed. A higher offset covers stronger move-

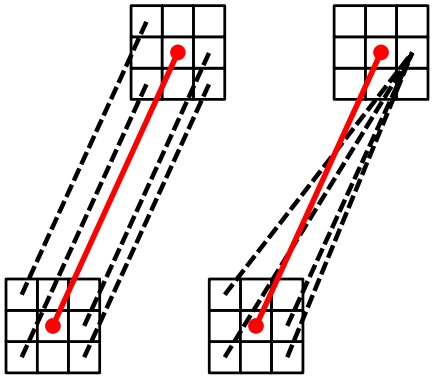


Figure 2: Generating edge samples (dotted) for a given model edge on a 3x3 window.

Figure 3: Sampled edges from the model in 2D image space. Only some random edges are shown for better visibility.

ments of the line features because more lines are sampled at greater distances and with wider angles, but this leads to a decrease in performance, especially when using occlusion queries. From the model knowledge the information about the depth of the 3D edge can be used to improve the sample edge generation. Movements far away from the camera lead to smaller shifting in pixel space while the same movement next to the camera is expressed in a large shift in pixel space. Knowing the depths of start and end point from the model edge, we can define different sizes for the sampling windows for both points, i.e. perspectively dependent generation of sample edges. If the depths of both points differ more than a threshold, for the nearer point more sample points are generated than for the farther point. This reduces the total number of sample edges to be rendered.

3.3 Occlusion Query Management

Retrieving the pixel count from the matching shader by occlusion queries affords a management process that can handle multiple queries to be executed fast. We have a list of model edges to be rendered and for each one a separate occlusion query has to be run. But the

graphics hardware limits the number of queries that can efficiently return a result in sequence. Trying to retrieve the counter result immediately after each query has finished would stall the CPU [Fer04].

While there are more edges to be rendered than queries can be executed, the task has to be splitted in several passes. A set of n maximal queries is created. A part of the model edges can be rendered until the maximum number of n available queries is reached. Then the results of all n queries have to be retrieved before a new block of n queries can be started for the remaining edges. The result with the highest count is stored. Alternatively an ordered list of the results can be created for better comparing of the results. The absolute number of query calls is also counted and the process finishes, when all edges have been drawn (See listing 2).

```
create n query objects
generate sample edges
enable shader
load canny texture
disable color and depth buffer write
while( query count != number of edges ){
    for n queries{
            start query
            render edge
            end query
            query count++
    }
    for n queries{
            retrieve result
            if query result > last query
                    save result
    }
}
enable color and depth buffer write
disable shader
```

Listing 2: Using managed occlusion queries.

3.4 Advanced Texture Read-Back

The results showed that using a simple shader with occlusion queries does not perform very well with large sets of edges to match (see section 4). Therefore, we developed a more sophisticated shader for matching a significant amount of edges in short time by extending our first shader approach. It is based on texture readback incorporating the sample edge generation. Opposing to the occlusion query approach the sample edges are not precomputed on CPU. The generation and computation of the sample edges is entirely transferred to graphics hardware. Thus the number of render passes is reduced to the number of model edges, instead of rendering each sample edge in its own pass. Further this solves the problem of stalling the CPU while waiting for the occlusion query result. The pixel count of all sample edges belonging to one model edge can be retrieved with one texture read-back.

In addition to the canny texture the pixel shader now gets the coordinates of projected 2D start and end point of the model edge and the offset for sample edge generation as input variables. As described below, the shader calculates new sample points in a window with the given offset around the start and end point of the model edge.

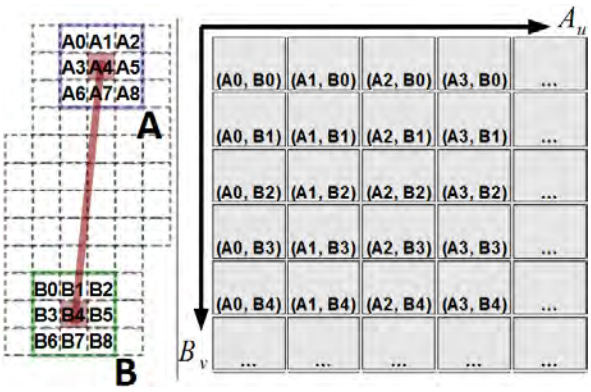


Figure 4: Shader target texture organization.

Each new start and end point is then connected to a sample edge. This is done by the shader performing the Bresenham line algorithm [Bre65a] between every start and end point generated. The pixel coordinates resulting from the line calculation are used to search for corresponding pixels on the canny-texture. The number of counted pixels is written as output value on the render-target texture. One render-target texture can store all counter results of the sample edges generated for one model edge. The texture has the size of all possible sample edges, e.g. when 9 sample points are generated in a 3x3 window for every start and endpoint, 81 sample edges are checked and this number of results has to be stored in the texture. Thus the texture must have size 9x9 for 81 entries.

Figure 4 shows the organization of the texture. For the start points A and end points B every column and its u-coordinate correspond to one start point in the sample window and every row and its v-coordinate correspond to one end point. Every pixel in the texture is now addressed for the result of one sample edge. The pixel shader is aware of the texture coordinate (u,v) it is going to write its value to, so it can use this information to apply an offset to the start and end point of the input model edge to generate the sample points. Thus, each pixel shader call calculates one sample edge depending on its writing position as follows.

Subtracting the offset from the x and y coordinates of the model edge start point A gives us the position of the first start point A0 with the lowest coordinates in the

sample window. From that point, adding the modulo of the u coordinate and window width s to the x coordinate and adding the division of the u coordinate by the window width s to the y coordinate results in the new sample start point:

$$s = 2*offset+1$$
$$sampleStartX = modelStartX - offset + (u \% s)$$
$$sampleStartY = modelStartY - offset + (u / s)$$
$$sampleEndX = modelEndX - offset + (v \% s)$$
$$sampleEndY = modelEndY - offset + (v / s)$$

Doing the same for the end point B and v coordinate returns the corresponding end point. Between these the Bresenham line will be calculated by the shader and the pixel count is written to the current position.

After the shader run the texture is read-back to CPU and the maximum value is determined by comparing all pixel values. From the pixel position (u,v) of the maximum and the offset applied, the pixel coordinates of the resulting sample edge can now be identified by the formulas shown above. It is also possible to use parallel reduction as described in [Fer04a] to directly obtain the maximum on the texture instead of using the CPU. Advanced computation of the resulting texture, like applying a non-maximum suppression leads to further quality criteria beneath the length of the matching line. The counting results of parallel edges are ordered diagonally on the target texture, which enables a quick check for this second quality criterion, e.g. given the case of figure 4 having a maximum at pixel position (2,2) and one or more significant high counts on one of the pixels in the diagonal from (0,0) to (8,8) this shows the existence of at least one parallel line with similar length and may lead to rejection of this match.

3.5 Optical Flow Support

Strong shaking of the camera may introduce a high level of motion blur. Due to the limited search area defined by the generated sample edges, the correct matching and subsequently the tracking may be lost when the image line is shifted too far. To prevent this, strong movements can be detected by estimation of optical flow [Bea95]. Calculating optical flow predicts the displacement of pixels in between two frames of an image sequence. The movement of the distinct start and end points of each projected model edge between the last and the current camera frame can be determined using a sparse optical flow function from OpenCV [Ocv11] that accepts an array of feature points as input. The predicted new start and end point positions corrected by optical flow are then used for sample edge generation. Matching with optical flow support allows reducing the distribution of sample edges because the matching can be performed in a smaller region. Using fewer sample edges leads to faster computation time. Another method to overcome motion blur is using additional inertial sensors [Rei06] to estimate rapid camera motion.

4 RESULTS

We tested our shader-based matching approach by tracking simple and complex objects on indoor and outdoor scenes (figure 5). As camera input we used video streams at a resolution of 640x480 pixels with varying lighting conditions. The canny filter applied to the camera images is taken from the OpenCV [Ocv11] implementation with standard parameters (threshold1 = 50, threshold2 = 200, aperture = 3).

The initial camera pose is assumed to be roughly known at the start of the sequence, which is a prerequisite for model-based tracking. This may be done by manually aligning the model in the camera frame. The intrinsic parameters of the video camera delivering the input stream are gained from previous calibration. Models of the tracking scenes are available and from these lists of the model edges are built to be used for the matching process. Each model edge is projected from the current camera pose and the matching shader returns the corresponding image line. For the computation of the new camera pose from the line correspondences we use a non-linear Levenberg-Marquardt optimization.

Figure 5: Test scenes (video and rendered model).

We compared our shader approach to two other distance-based matching methods. One method is to parameterize the binary canny image by a Hough transform [Dud72]. The model edge is projected into the image plane and a window around this edge defines a region of interest where the Hough transform is run. The output is a list of straight image lines defined by the parameters of line angle to the y-axis and line distance to the image origin. These can be directly compared to the parameters of the corresponding

model edge. However, matching in two dimensional parameter space proves to be very unstable. Neither length nor line similarity is judged this way, so we did not further consider this approach. Measuring the distance in image space can be done by projection of the start and end point of the model edge to the straight image line found by the Hough transform. The problem is the high dependency of the results on the chosen parameters of the transform. Possible matches are extremely ambiguous and lead to jittering in the estimated pose. At strong motion some matches completely fail.

Another popular approach is to set control points along the model edge and search for strong image gradients on orthogonal lines through these control points. We used an implementation from [Vis11] for our tests. The pose becomes more stable, but movements or interruptions in the image lines corrupt the matching result. Generally, these distance measures in image space can lead to stable camera pose estimation when the camera movement is slow and smooth enough, which is the case in the controlled indoor test scenario (figure 5 top, middle). However, at fast camera movements inducing motion blur the matching fails. We will show this on examples of the outdoor scene (figure 5 bottom).

Our matching shader has proven to deliver good matching results with minimal error even in the worst scenario of a video captured with a strongly shaking hand camera. The following figures illustrate the results of three matching approaches on a test sequence after the occurrence of strong motion. The motion blur occurs for duration of 6 frames while the image content is shifted over 80 pixels in this time. We regard the matching error in the frame right after this strong motion. Figure 6 shows the shift of the image within the 6 frames and the sub-picture the moment of strongest motion blur in the video sequence which disturbs the canny image to a large extent. Image lines are only partly visible and hard to handle by the matching methods.

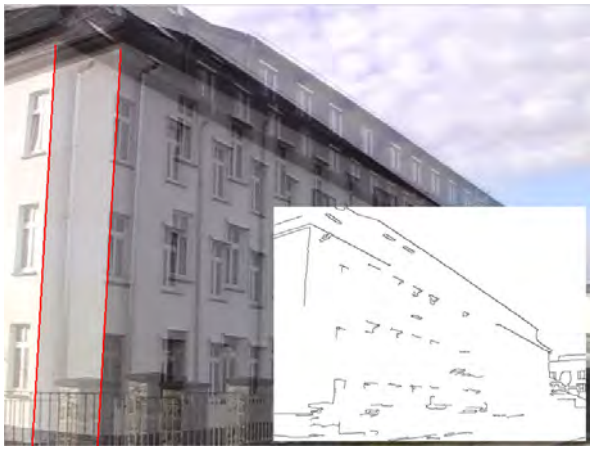


Figure 6: Image shift with strong motion blur.

The line projection approach (figure 7) obviously gets distracted by the parallel pipe next to the house corner, which is an ambiguous feature. The error ends up with a maximum displacement of 37 pixels. The matching with orthogonal search (figure 8) is more precise concerning ambiguities but also gets disturbed by the motion blur up to an error of 24 pixels. The matching shader approach with optical flow support (figure 9) overcomes the blur and results in an error of 3 pixel displacement.

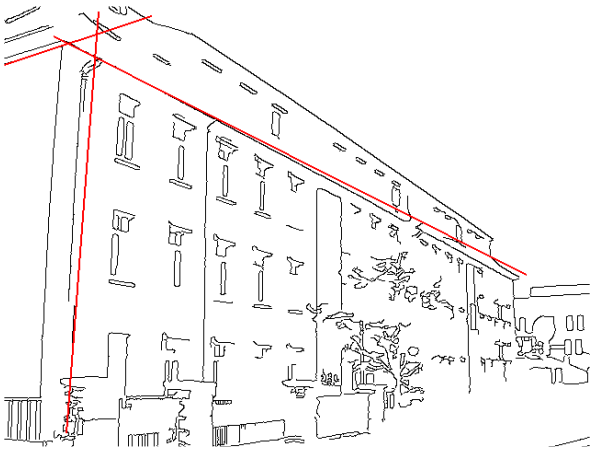


Figure 7: Line projection results after motion.

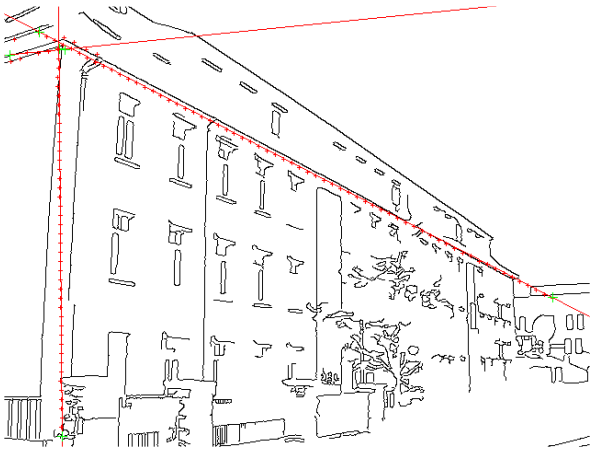


Figure 8: Orthogonal search results after motion.

In figure 10 the moment of strongest motion blur in the outdoor scene can be seen together with the resulting pose estimation overlay from the matches proposed by our method. Concerning the parameters, in our tests we found a good window size for the generation of sample edges at 4x4 with supporting optical flow. Without optical flow the best trade-off between computation time and matching quality could be reached with generating sample edges at 7x7 windows. The best threshold for rejection of the image line length as match is $\frac{3}{4}$ of the model line length.

Table 1 lists the average computation time in milliseconds of the components for our matching approach on

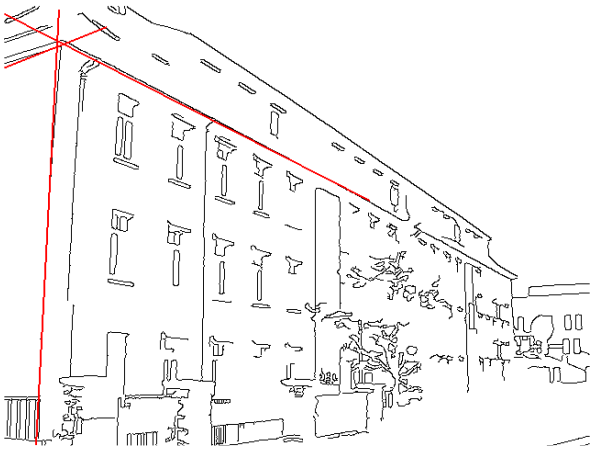


Figure 9: Matching shader results after motion.

Figure 10: Correct camera pose computation at strong motion blur (top) and for other scenes.

a Intel Core2Duo 3.2GHz with nVidia GeForce GTX 285. The canny filter and optical flow calculation step are called only once for a new frame, independently of the number of edges to match. Although time consumption for the canny filter is not too high, this additional step can be reduced by implementing gradient search inside the shader instead.

Next, times for the occlusion query approach and the texture read-back variant are compared. The number in brackets names the amount of edges to render. Obviously using a query executes fast only when a very little number of edges is used. But enlarging the number of model edges the texture approach quickly outperforms the query usage. Overall, real-time capability for tracking is ensured, however the implementation is not yet optimized.

Canny filter	5 ms
Optical flow	4 ms
Occlusion query	97 ms (11), 54(7), 7(2)
Texture read-back	18 ms (11), 14,(7), 7(2)

Table 1: Computation time.

5 CONCLUSION

We presented a shader approach for matching corresponding image lines to model edges in a model-based tracking scenario. The knowledge about the model is used to improve the matching and to define criteria for match selection. For a given model edge based on the last pose several sample edges are generated and rendered with a matching shader. The shader counts underlying pixels of a canny-filtered camera input image at the position of the edges. The image line with highest accordance to criteria of length and distance is chosen as match. This procedure delivers good matching and results in a correct camera pose estimation even at occurrence of strong motion blur.

We compared two methods to realize the matching shader. Using a simple counting shader and occlusion queries to retrieve the pixel count result is straightforward but significantly lowers the frame rate when many sample edges are generated because each render pass. The more sophisticated way is to read-back a texture value which can be done quite fast. The whole process of sample edge generation can be transferred into the shader, so a render pass is only called once per model edge instead for each sample edge.

Although we used straight lines from our testing models, the work is not limited to this type of feature. The counting shader can be extended to run on other renderable structures as well, like circles, curves or NURBS. For this purpose the sample generation algorithm has to be adapted to the wanted structure to match. A further advancement could be the integration of gradient calculation into the shader. This would save the canny preprocessing step to the camera image. Additionally, when calculating the gradient, the gradient orientation is also known. This could be used by the matching shader to count those pixels only, which have the same gradient direction and thus belong to the same line.

6 ACKNOWLEDGMENTS

This work was supported by grant no. MU 2783/3-1 of the German Research Foundation (DFG). We also want

to thank our former colleague Niklas Henrich for his support on GPU programming and Carsten Neumann from University of Louisiana at Lafayette for technical discussions.

7 REFERENCES

[Bay08] H. Bay, A. Ess, T. Tuytelaars and L. Van Gool. SURF: Speeded Up Robust Features. Computer Vision and Image Understanding (CVIU), 110(3), pp. 346–359, 2008.

[Bea95] S.S. Beauchemin and J. L. Barron. The computation of optical flow. ACM Computing Surveys, 27(3), pp. 433-466, 1995.

[Bou89] P. Bouthemy. A Maximum Likelihood Framework for Determining Moving Edges. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, pp. 499-511, 1989.

[Bra11] A.K. Braun and S. Mueller. GPU-assisted 3D Pose Estimation Under Realistic Illumination. 18th WSCG International Conference on Computer Graphics, Visualization and Computer Vision, Plzen, Czech Republic, 2011.

[Bre65] J.E. Bresenham. Algorithm for Computer Control of a Digital Plotter. IBM Systems Journal, 4(1) , pp. 25-30, 1965.

[Bro12] J.A. Brown and D.W. Capson. A Framework for 3D Model-Based Visual Tracking Using a GPU-Accelerated Particle Filter. IEEE Transactions on Visualization and Computer Graphics, 18, pp. 68-80, 2012.

[Can86] J. Canny. A Computational Approach To Edge Detection. IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6), pp. 679-698, 1986.

[Com03] A.I. Comport, E. Marchand and F. Chaumette. A Real-Time Tracker for Markerless Augmented Reality. ACM/IEEE Int. Symp. on Mixed and Augmented Reality, pp36-45, Tokyo, Japan, 2003.

[Dru02] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7), pp. 932-946, 2002.

[Dud72] R.O. Duda and P.E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures. Communications of the ACM, 15(1), pp. 11-15, 1972.

[Fer04] R. Fernando. GPU Gems. Programming Techniques, Tips, and Tricks for Real-Time Graphics. Addison-Wesley, Longman, Amsterdam, 2004

[Har90] C. Harris and C. Stennet. RAPID - A Video Rate Object Tracker. In Proc. British Machine Vision Conference, pp. 73-77, Oxford, UK , 1990.

[Kle06] G. Klein and D. Murray. Full-3D Edge Tracking with a Particle Filter. Proc. British Machine Vision Conference (BMVC'06), 3, pp. 1119-1128, 2006.

[Kle07] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. ACM/IEEE Int. Symp. on Mixed and Augmented Reality, pp. 225-234, Nara, Japan, 2007.

[Low91] D.G. Lowe. Fitting Parameterized Three-Dimensional Models to Images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(5), 1991.

[Low92] D.G. Lowe. Robust model-based motion tracking through the integration of search and estimation. International Journal of Computer Vision, 8(2), pp. 113-122, 1992.

[Low99] D.G. Lowe. Object Recognition From Local Scale-Invariant Features. International Conference on Computer Vision, pp. 1150-1157, Corfu, Greece, 1999.

[Ocv11] OpenCV library, version 2.3.1, taken from http://opencv.willowgarage.com/wiki/

[Rei06] G. Reitmayr and T.W. Drummond. Going out: Robust Tracking for Outdoor Augmented Reality. International Symposium on Mixed and Augmented Reality (ISMAR06), pp. 109-118, 2006

[Sch09] M. Schumann, S. Achilles and S. Mueller. Analysis by Synthesis Techniques for Markerless Tracking. Virtuelle und Erweiterte Realität, 6. Workshop der GI Fachgruppe VR/AR, Braunschweig, Germany, 2009.

[Shi94] J. Shi and C. Tomasi. Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition, pp. 593-600, 1994.

[Sin06] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. GPUbased video feature tracking and matching. Workshop on Edge Computing Using New Commodity Architectures (EDGE 2006), 2006.

[Vac04] L. Vacchetti, V. Lepetit and P. Fua. Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking. ACM/IEEE Int. Symp. on Mixed and Augmented Reality, pp. 48-57, Arlington, USA, 2004.

[Vis11] ViSP Visual Servoing Platform library, version 2.6.1, taken from http://www.irisa.fr/lagadic/visp/visp.html

[Wue05] H. Wuest, F. Vial and D. Stricker. Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality. ACM/IEEE Int. Symp. on Mixed and Augmented Reality, pp. 62-69, Santa Barbara, USA, 2005.

[Wue07] H. Wuest and D. Stricker. Tracking of Industrial Objects by Using CAD Models. Journal of Virtual Reality and Broadcasting, 4(1), 2007.

Active Segmentation in 3D using Kinect Sensor

Zoltan Tomori
Inst. of Experimental Physics
Slovak Academy of Sciences
Watsonova 47
040 01 Kosice, Slovakia
tomori@saske.sk

Radoslav Gargalik
Inst. of Computer Science
P.J. Safarik University
Jesenna 5,
040 01 Kosice, Slovakia
radoslavgargalik@gmail.com

Igor Hrmo
Inst. of Experimental Physics
Slovak Academy of Sciences
Watsonova 47
040 01 Kosice, Slovakia
hrmo@saske.sk

ABSTRACT

The combination of color image and depth map significantly improves the segmentation. The Kinect sensor with pan/tilt motorized movement captures both images and segments them separately by the Grab Cut method. The resulting contours are converted to polar coordinates. After the floor plane detection, corresponding "depth" and "color" contours are combined such that the importance of depth /color information is proportional to the distance from the floor. The segmentation is followed by the extraction of simple scale invariant features like color components and height/width ratio. Subsequently, features are used to train Normal Bayes Classifier. The algorithm was tested on a set of simple objects (mugs) on the table.

Keywords

Active segmentation, Kinect, Depth map, RGBD image

1. INTRODUCTION

Segmentation in 3D is a critical problem in many areas of computer vision. The information about the depth can be obtained by various methods like stereo vision, moving camera or object, defocusing, structured light or comparison with known geometrical model [Mir04]. Kinect - a low-cost 3D sensor for gaming console was launched in November 2010 and achieved big commercial success. Support for programmers appeared shortly after it (Microsoft Kinect SDK, OpenNI, OpenKinect, Freenect etc.). A more comprehensive source of information about Kinect hardware and programming was published only recently - e.g. [Web12].

Kinect provides 3D information which can be easily retrieved (color, depth, points cloud in real distance units). Its depth sensor consists of infrared transmitter/camera system. The transmitter projects small dots (speckles) on the surrounding scene and the IR camera acquires image and compares their position with the reference one. The depth is then calculated from the displacement of the individual speckles. The color + depth images are acquired approximately at the same moment and after their alignment they can be considered as one RGBD image.

The concept of active segmentation was inspired by the biological visual system where the object of interest is segmented with high resolution by the fovea, while the rest of the scene is captured in lower resolution on the periphery of retina [Mis09]. Another important concept is the "contact boundary" introduced in [Mis11] where the boundary pixels touching the surface (floor, desk) are distinguished from the remaining ones. Contact boundary is important in segmentation based on the combination of color image and depth map, both provided by Kinect.

2. ACQUISITION

Mechanical Construction

For testing purposes, we constructed a motorized equipment and we named it KATE (Kinect Active Tracking Equipment) - see Figure 1a). It consists of the Kinect sensor attached to the turntable driven by a precise stepper motor (Intelligent Motion Systems, USA). This allows horizontal movement around its vertical axis (pan) with the resolution 51200 steps per 360 degrees. For the vertical movement (tilt) we exploited the built-in Kinect stepper motor controlled via the same USB port as cameras. This solution is simple but it has a poor resolution in range <-31, 31> degrees. Another problem is that the tilt value is related to the absolute horizontal position measured automatically by Kinect accelerometer. However, this solution is sufficient for testing purposes.

a) **b)**

Figure 1. a) KATE (Kinect Active Tracking Equipment) b) Object as seen by Kinect. Central rectangle represents the region of interest (fovea) where the active segmentation is performed.

Calibration and tracking

Camera calibration based on the pinhole camera model is necessary in applications like objects reconstruction from multiple views. As Kinect consists of 2 cameras (RGB and IR), more complicated model is required [Paj11]. However, initial calibration included in OpenNI library is sufficient for segmentation and tracking purposes. The depth values correlate well with real values, the correspondence between the depth (Z) and XY dimensions can be easily corrected by scale constants.

We developed a supervised calibration function adjusting the scale in both horizontal and vertical directions. KATE watches the given scene displaying live image along with the cross in the image center $p_c(x,y)$ - see Figure 1b). The operator clicks a position $p_1(x,y)$ representing the desired future position of the image center. From p_c and p_1 values in pixels we can find corresponding 3D points $P_c=(0, 0, Z_0)$ and $P_1=(X_1, Y_1, Z_1)$ in real units (meters). Using simple trigonometry we obtain pan angle as $arcsin(X_1/Z_1)$ and tilt as $arcsin(Y_1/Z_1)$.

Calculated angles are converted to the stepper motor units and both motors move the given number of steps. In ideal case, the new central position $p_c(x,y)$ should show the same place of the scene as the $p_1(x,y)$ before the movement. If it is not the case, the scale factor can be adjusted. The calibration function allows manual control of the motors. The number of correcting steps is recalculated to the new scale factors for both pan and tilt values.

We created a face tracking system to test KATE [Sen12]. The new position of motors is not given by the click as described in the previous section, but it is determined by the face bounding rectangle (see Figure 2). Face detector is based on the Haar cascade classifier included in the OpenCV library. If a face is recognized (in the predefined depth range) then the displacement between the face rectangle and the center of the image is calculated. Pan/tilt motors correct the Kinect position trying to keep the face rectangle in the image center.

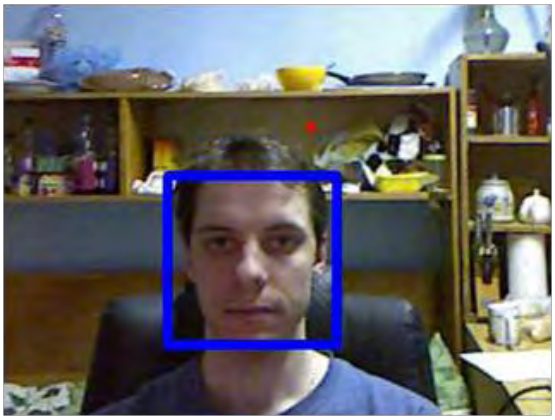


Figure 2. Face tracking. Kinect detects face by using Haar classifier included in OpenCV library. In the next step, pan/tilt motors adjust Kinect position such that the face rectangle is in the center of image.

Preprocessing

The depth image contains a lot of artifacts resulting from the depth measurement principle. Shadows-like defects appear in places visible from the depth camera but not illuminated by IR projector.

We exploited the "Inpaint" method described in [Tel04], which recovers the missing depth information. Implementation of this method is easy as it is included in OpenCV library.

3. PROCESSING

Processing consists of the following sequence of steps: plane detection, finding volume of interest and segmentation.

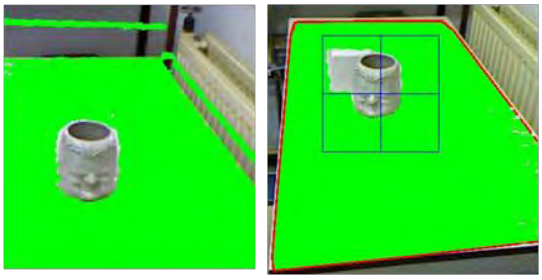
Floor Plane Detection and Region of Interest (ROI)

One of the basic operations in computer vision is the detection of the plane where the segmented objects are standing on (floor, table top). The detection is based on the popular RANSAC algorithm which finds the plane representing the input points cloud. The plane is determined by the equation

$$ax + by + cx + d = 0 \quad \textbf{(1)}$$

where $[a,b,c]$ is the normal vector and d is the distance from the origin.

Although RANSAC eliminates the influence of outliers in principle, it is possible to improve the plane detection by filtering the points that are definitely outliers. There are a lot of RANSAC modifications performing this task [Chu03]. It is possible to iterate plane detection and calculation of the volume of interest (described in the next section). Data for the next plane detection create only voxels from the volume of interest.



a) **b)**

Figure 3. Floor plane detection. a) RANSAC detects not only table-top voxels but also unwanted strips of voxels on the walls b) Region of interest represented by the top of the table (green) outlined by polygon (red).

Figure 3a) shows the detected plane (green pixels). As can be seen there, plane consists of the table top area as well as some undesirable green strips on the walls. We filtered them as follows:

- Erosion filter isolates table top from the connected strips (like this on the top right corner of table).
- Table top contour is found as the largest contour which includes center of image (fixation point).
- Convex hull function eliminates local discontinuities and gives the polygon outlining the table top (red in Figure 3b).

Finding the Volume of Interest (VOI)

The majority of applications focus on a limited space above the floor or the table-top. The bottom of this space is represented by the ROI detected in the previous step. Its top is given by the maximal expected height of our objects.

Limits from sides (walls, furniture) can be found by an algorithm based on a simplified model of the scene assuming that side limits are higher than segmented objects.

1) Project all voxels higher than the top limit to the floor plane.
2) Find the polygon with maximal area that does not include projected points. The number of vertices is given in advance.
3) Side limits of our volume of interest are created by planes perpendicular to the floor crossing the polygon sides.
4) Project voxels lying between the top and the bottom limits. If there are points falling inside the polygon and connected with its sides, algorithm returns to step 2.

Segmentation by Grab Cut

The algorithm exploited in our experiments is based on the idea of active segmentation briefly explained in the introduction. From a lot of possible segmentation methods we focused on these which are based on the classification of pixels inside a region of interest (ROI) and the minimization of an energy function. We assume that the segmented object is placed inside the square frame which is our initial region of interest.

GrabCut [Rot04] is a very popular segmentation algorithm based on the Gaussian Mixture Model and energy minimization. All pixels outside the ROI are labeled as background ones, pixels inside ROI are labeled according to the energy function consisting of regional and boundary terms. Briefly speaking, regional term reflects the likelihood that a given label is appropriate for the given pixel and the boundary term reflects how easily the label can expand to its neighborhood. Assigning a boundary label to a pixel inside a homogeneous region is penalized. The strength of N-link (the link between pixels *m* and *n*) is calculated

$$N(m,n) = \frac{\gamma}{d(m,n)} \exp\left(-\beta \left\| Z_m - Z_n \right\|^2\right) \quad \textbf{(2)}$$

$$\beta = \frac{1}{2\left\langle \left\| Z_m - Z_n \right\|^2 \right\rangle} \quad \textbf{(3)}$$

where Z_m is the color of the pixel *m*, constant γ has recommended value = 50, *d(m,n)* is the unit distance (1 for horizontal and vertical neighbors and *sqrt(2)* for diagonal ones). Value of β is the average inverse difference value calculated in advance.

Combined Segmentation

Kinect generates two images reflecting the same scene - color image and the depth map. Figure 4 shows the problem of color image segmentation if the other object with similar color is behind our object of interest. On the other hand, the depth map segmentation has troubles with parts near the floor where the depth of object and the background are almost the same.

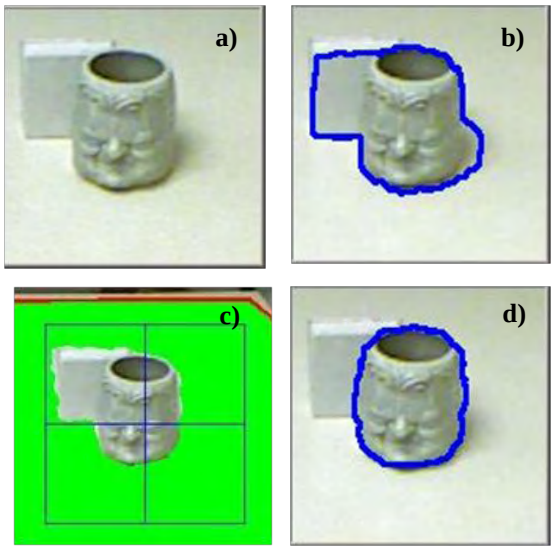


Figure 4. a) Overlapping objects with shadow on the right side. b) Segmentation by the GrabCut method exploiting only the color image. c) Floor plane detection (green pixels) d) Segmentation of depth image masked by the floor plane.

Above mentioned problems can be solved combining information from both - the depth map and color image. Several authors [Kar10], [Mut10] combine color and depth information. In [Sil11] a model was proposed which modifies computation of N-link as follows:

$$N(m,n) = kN_C(m,n) + (1-k)N_D(m,n) \quad \textbf{(4)}$$

where N_C and N_D are N-links from color image and depth image respectively and k controls the influence of both links to the final N-link calculation (e.g. 80% color and 20% depth). Modified N-link calculation is incorporated into the energy function optimized by graph cut method.

Distance Dependent Segmentation

Our approach is based on the assumption that the value of k from **(4)** is not constant but depends on the distance from the floor. In standard situations, the object standing on the floor is captured by Kinect under tilt angle. The top part of the object has a much higher contrast of the depth image than its bottom (Figure 5b). We combine the results of contours segmented from both color and depth images.

The initial step is a conversion of both contours from Cartesian to polar coordinates using the center of the image as the origin. The x-axis corresponds to the angle in the clockwise direction starting from the position "3" on the clock, y-axis is magnitude. The blue curve in Figure 5c corresponds to the contour from the color image and the green one to the depth image, the resulted red curve is their combination. After its conversion back to the Cartesian coordinates we can obtain a contour shown in Figure 5d.

Several alternatives exist how to change the color/depth influence, depending on the type and configuration of the objects on the scene.

a) Piecewise combination of segmented curves is the simplest method. The depth image contour representing the contact boundary is replaced by the corresponding part of the color image contour in a pre-selected interval. For instance, an interval <45,135> degrees represents the bottom of an object where a low contrast of depth image is expected. Selection of the end points of the interval should respect the continuity of resulted curve. Good candidates are intersections of the both curves.

b) Derivation of k on the distance from the plane using normal vector (1). As Kinect gives *(x,y,z)* values in the camera coordinate system, the calculation of the distance from the plane is straightforward.

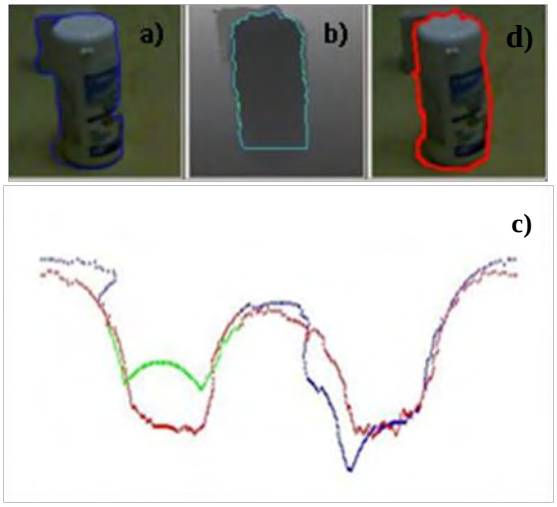


Figure 5. Segmentation by combined GrabCut method. a) color image b) depth map c) contours in polar coordinates d) result of segmentation

Classification of segmented objects

The successful segmentation is often the crucial step in computer vision (e.g. in objects recognition based on machine learning principles). We tested our system to recognize several simple objects (like mugs on the table). We exploited supervised learning based on the Normal Bayes Classifier. System segmented each object on the table and found the bounding rectangle of its contour as well as the average color. Feature vector consisting of 4 components (R, G, B color components and height/width ratio of bounding

rectangle) was used in the supervised learning stage taking cca 5 seconds. After the learning, the system was able to recognize object and display its label (Figure 6).

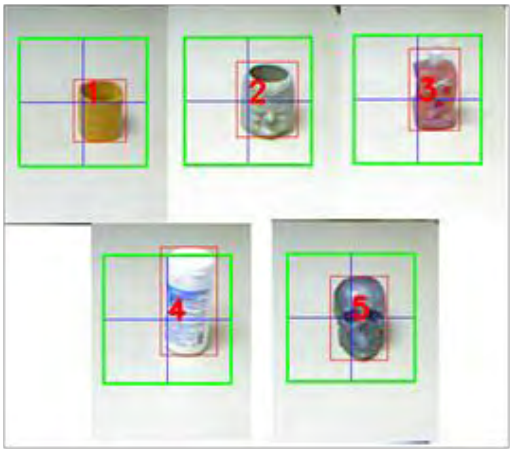


Figure 6. Classification of segmented objects using Normal Bayes Classifier based on the feature vector created by RGB color components and height/width ratio.

4. SUMMARY

Motorized pan/tilt Kinect system was constructed for the acquisition of color and depth images. This system tracks the object of interest keeping it in the center of image (tested with face tracking based on the Haar cascade classifier included in OpenCV library).

We applied Grab Cut method to segment both color and depth images using the image center as the fixation point. We transformed contours into the polar coordinates and combined them. The weights controlling the importance of color/depth edges was dependent on the distance from the floor detected by RANSAC method. This approach significantly improved segmentation near the floor as well as in partially overlapping objects. Segmented contours were used for the features extraction (R, G, B color components and height/width ratio). We used this features vectors for supervised training of Normal Bayes Classifier and for the classification of simple objects like mugs on the table.

Future work

This communication paper reflects our experiments with Kinect as the initial stage of the project oriented to application of Natural User Interface. We plan to exploit RGBD images for wider group of problems like active segmentation, tracking and control of specific devices. Growing number of papers combining color + depth along with the progress in sensors hardware make this research area very promising.

5. ACKNOWLEDGMENTS

This work was supported by the Slovak research grant agencies APVV (Project No. 0526-11) and VEGA (Project No. 2/0191/11).

6. REFERENCES

[Chu03] Chum, O. Matas, J. and Kittler, J. Locally Optimized RANSAC, Lecture Notes in Computer Sciences, 2781, pp. 236-243, 2003.

[Kar10] Karthikeyan, V. Anil, A. and Ebroul, I. GrabcutD: improved grabcut using depth information. Proc. ACM workshop on Surreal media and virtual cloning, Firenze, Italy, pp. 57-62 , 2010.

[Mir04] Mirzabaki, M. Depth Detection Through Interpolation Functions: A New Method. Proc. WSCG, Plzen, Czech Rep., pp. 105-108, 2004.

[Mis11] Mishra, A. and Aloimonos, Y. Visual Segmentation of "Simple" Objects for Robots. Proc. Robotics Science and Systems conference (RSS), Los Angeles, June 27 - July 1, 2011. http://www.umiacs.umd.edu/~mishraka

[Mis12] Mishra, A. K. Aloimonos, Y. Cheong, L. F. and Kassim, A. A. Active Visual Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34, pp. 639-653, 2012.

[Mut10] Mutto, C. D. Zanuttigh, P. and Cortelazzo, G. M. Scene Segmentation by Color and Depth Information and its Applications. Proc. Streaming Day, Udine, 2010.

[Paj11] Pajdla, T. Smisek, J. and Jancosek, M. 3D with Kinect. Proc. of the 1st IEEE Workshop on Consumer Depth Cameras for Computer Vision, Barcelona, Spain, pp. 1154-1160, 2011.

[Rot04] Rother, C. Kolmogorov, V. and Blake, A. "GrabCut" - Interactive foreground extraction using iterated graph cuts. ACM Transactions on Graphics, 23, pp. 309-314, 2004.

[Sen12] Senaj, M. OpenCV library in computer vision applications in robotics, Master's thesis, FEI, Technical University of Kosice, 2012.

[Sil11] Silberman N. and Fergus, R. Indoor Scene Segmentation using a Structured Light Sensor.Proc. Int. Conf. on Computer Vision - Workshop on 3D Representation and Recognition, Barcelona, 2011.

[Tel04] Telea, A. An image inpainting technique based on the fast marching method. Journal of Graphics Tools, 9, pp. 23-34, 2004.

[Web12] Webb, J. and Ashley, J. Beginning Kinect Programming with the Microsoft Kinect SDK. Apress, (ISBN 978-1-4302-4104-1), 2012.

Using Game Engine Technology for Virtual Environment Teamwork Training

Stefan Marks
Auckland University of Technology, New Zealand
stefan.marks.ac@gmail.com

John Windsor
The University of Auckland, New Zealand
j.windsor@auckland.ac.nz

Burkhard Wünsche
The University of Auckland, New Zealand
b.wuensche@auckland.ac.nz

ABSTRACT

The use of virtual environments (VE) for teaching and training is increasing rapidly. A particular popular medium for implementing such applications are game engines. However, just changing game content is usually insufficient for creating effective training and teaching scenarios. In this paper, we discuss how the design of a VE can be changed to adapt it to new use cases. We explain how new interaction principles can be added to a game engine by presenting technologies for integrating a webcam for head tracking. This enables head-coupled perspective as an intuitive view control and head gestures that are mapped onto the user's avatar in the virtual environment. We also explain how the simulation can be connected to behavioural study software in order to simplify user study evaluation. Finally we list problems and solutions when utilising the free Source Engine Software Development Kit to design such a virtual environment. We evaluate our design, present a virtual surgery teamwork training scenario created with it, and summarize user study results demonstrating the usefulness of our extensions.

Keywords: Serious Game, Source Engine, Medical Teamwork Training, Head Tracking, Non-Verbal Communication, Head-Coupled Perspective

1 INTRODUCTION

In recent years, virtual environments (VEs) have become increasingly popular due to technological advances in graphics and user interfaces [MSL+09]. One of the many valuable uses of VEs is teamwork training. The members of a team can be located wherever it is most convenient for them (e.g., at home) and solve a simulated task in the VE collaboratively, without physically having to travel to a common simulation facility. Medical schools have realised this advantage and, for example, created numerous medical simulations within Second Life or similar VEs [DPH+09].

To implement a VE, the developer has to choose between three possibilities:

1. To use a completely implemented commercial or free VE solution like Second Life [Lin10]. This has the advantage of being able to completely focus on content creation instead of having to deal with technical implementation questions and problems. However, the disadvantage is that these frameworks cannot easily be extended with additional functionality required for a specific simulation scenario.
2. To build a VE from scratch. This enables complete freedom in the design and usability of the VE, but significantly extends development time.
3. To use a simulation framework that can be flexibly extended to account for special design requirements, but already provides a solid foundation of functionality to achieve a quick working prototype.

Whereas the first two options are located at the opposite extremes of the spectrum, the last option is located between these extremes in terms of development flexibility and rapid prototyping. A game engine, the underlying component of computer games, can be used as such a framework. This is the principle behind "serious games": To use the technology of computer games, e.g., graphics, sound, physical simulation, multi-user support, but to replace and adapt the original content to build "serious" applications, e.g., for education, training, or simulation.

The literature provides several examples of studies with simulation environments based on game engines, e.g., [TSHW08], [MRL+06], [ST09]. For an extended review of serious games, see [SJB07].

However, rarely does the reader find information discussing design options, the advantages and disadvantages of tools such as game engines, and how to use them effectively and integrate new functionalities. This makes it difficult for researchers to extend existing simulations or create new ones. One example of the few exceptions is the publication of Ritchie, Lindstrom, and

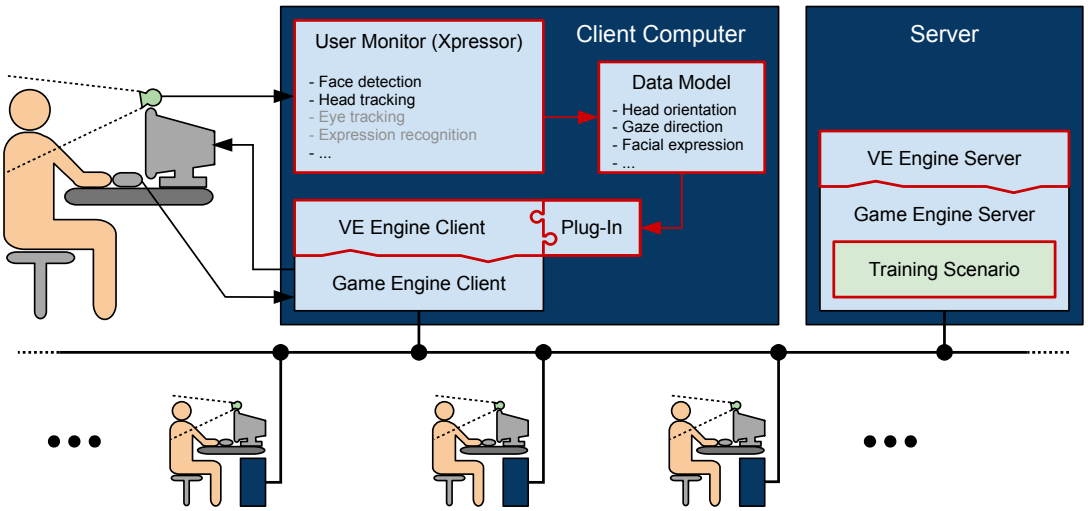


Figure 1: The functional blocks of the simulation framework

Duggan, where not only the used tools for the development process but also source code details are provided [RLD06].

We have created a VE for medical teamwork training which provides additional control mechanisms by using a webcam to capture the head movement of the user. This head movement is decomposed into the translational part which is used for head-coupled perspective (HCP), and the rotational part which is used to control head gestures of the user's avatar to convey non-verbal communication cues. The results of our user studies show that HCP improves the usability of the VE as it introduces an intuitive view control metaphor that even inexperienced users were able to master within seconds. In addition, tracking-based head gesture control of the avatar improved the perceived realism of the simulation [MWW11].

This paper provides insight into the design of game engines and their modification for advanced "serious games" applications. In particular we explain how new user interface devices can be integrated. In our discussions, we use the Source Engine [Val07] as an example, which, at the time of the survey, fulfilled most of our simulation requirements: good graphical and animation capabilities, availability of an Software Development Kit (SDK) and developer tools, and reliable synchronisation of physically simulated objects among multiple clients. The details of the selection process of a suitable engine can be found in [MWW07].

Section 2 presents the design of our VE framework. In Section 3, we describe the details of the implementation. A summary of the results of our user studies conducted with the framework are then presented and discussed in Section 4, and we finish with the conclusion in Section 5.

2 DESIGN

The goal of the research described in this paper is to utilise a game engine to implement a virtual environment for surgical teamwork training. An important component in teamwork is non-verbal communication, such as head gestures, which we capture with a webcam and then map onto avatars in the virtual environment. In addition we need an intuitive method for view control, since most surgical procedures require the surgeon to use both hands for instruments. We therefore decided to implement HCP using webcam input. HCP changes the view of the VE based on the movements of the user's head position in front of the monitor. The implementation of these additional hardware components and control metaphors is also part of this paper.

Realising our simulation scenario requires changing the game content, gameplay, and integration of webcam input to control game engine parameters. Figure 1 gives an overview of the architecture of the resulting system. The sections marked in red were developed, extended, or modified for the implementation. Figure 2 shows a screenshot of the final VE for teamwork training simulations.

The simulation is run on a central server that all users connect to with their client computers. The server as well as the clients run their part of the VE engine, being constructed on top of the Source Engine. It is important to distinguish between the terms "game engine" referring to components of the simulation framework that are parts on the Source Engine itself and therefore cannot be altered, and "VE engine" referring to components that are based on the Source SDK and have been altered to create a VE with new features and interactions.

The original game content on the server is replaced by the teamwork training scenario. This includes virtual

Figure 2: Screenshot of the final surgical teamwork simulator *MedVE* created from the original deathmatch game code

rooms, objects, instruments, sounds, textures, 3D models, etc.

On each client, an additional program, called *Xpressor*, is running, using the input from the webcam for tracking the user's head and face. The tracking information is sent in the form of a specific data model (see Section 3.3) to a plug-in of the VE engine. By using an external tracking program and the plug-in architecture, it is easily possible to exchange these components later with more advanced ones, without having to modify the actual VE engine.

The translational head tracking information is used to control the view "into" the VE. This so called head-coupled perspective (HCP) enables intuitive control, such as peeking around corners by moving the head sideways, or zooming in by moving the head closer to the monitor.

The rotational head tracking information is used to control the head rotation of the user's avatar. That way, other users in the VE can see head movement that is identical to the movement actually performed physically by the user, such as nodding, shaking, or rolling of the head.

In addition, data from face tracking can be used to detect facial expressions and transfer them onto the user's avatar. Bartlett et al [BLWM08], for example, present a system that recognises a large set of movements of facial keypoints, such as lip corners, eyebrows, or blinking. Using a simpler set of movements and keypoints, the authors of [QCM10] created a virtual mirror, where an avatar mimics smile, gaze and head direction, and an opened/closed/smiling mouth of the user in realtime. In our implementation, we use a non-commercial version of the face tracking library *faceAPI* which does *not* include the detection of facial expressions.

3 IMPLEMENTATION

In the following three sections, we will explain the implementation of the three important components of this framework: the virtual environment, the user monitor *Xpressor*, and the data model.

3.1 Virtual Environment

3.1.1 Steam Client

The modification of a game that utilises the Source Engine starts off with *Steam*, a client software of the manufacturer Valve, designed to enable the user to buy games online, download and install them, and to keep the games updated when bugfixes or extras are released.

3.1.2 Creating a New Project

The Source SDK is available to anybody who has purchased at least one game that utilises the Source Engine, e.g., *Half-Life 2*, *Team Fortress 2*, *Portal*. With the help of the *Steam* client, the SDK is easily downloaded and installed like any other game. The SDK gives a good starting point to experiment with the code and the game engine, and to start modifying certain aspects of the game. Several pages on the Valve Developer Community (VDC) website give additional hints and ideas, for example the "My First Mod" tutorial [Val10b].

3.1.3 Version Control

Directly after creating and compiling the modification project, we put the code under version control, using Subversion [The11], as described in [Val10e]. That way, we were able to update the code with changes that were applied to the SDK later in the development process.

3.1.4 Concepts of the Source Engine

Gaining an understanding of the design and functionality of the Source Engine was a very time-consuming process. At first sight, the developer website creates an impression of a thorough documentation. But when it comes down to details and specific questions, this documentation reveals large gaps and provides outdated or even contradictory information.

The majority of work necessary for understanding the Source Engine was to read through the provided code of the SDK, ignore several inconsistently implemented naming conventions, insert execution breakpoints, trace method calls through several class inheritance levels, and much more.

Good documentation and a well structured codebase is important for any game engine that is to be the foundation of a simulation. Without these prerequisites, a lot of time is spent on deciphering the inner workings of the underlying code or on figuring out how to achieve

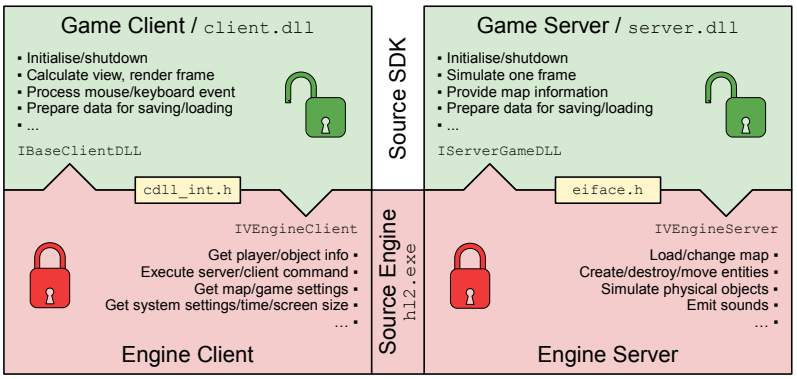


Figure 3: Boundaries between the Source SDK and the Source Engine

a certain functionality, instead of implementing the essential parts of the program.

In the following sections, we will present some of the major concepts of the Source Engine that played an important role in the development and modification phase.

3.1.5 *Game Engine/SDK Boundaries*

When a multi-user Source Engine game is started, four program parts are involved (see Figure 3):

- The game engine (`hl2.exe`) is executed, consisting of the server
- and the client part.

Depending on whether the user chooses to start a new game server or to connect to an existing game server, the engine then activates

- either the game server dynamic link library (DLL) `server.dll`
- or the game client DLL `client.dll`.

The game engine itself cannot be modified at all. No source code of the inner workings of the engine is given.

The SDK contains header files with interface definitions for the server (`eiface.h`) and the client part (`cdll_int.h`) of the engine. These interfaces provide access to very basic entity and resource management, and to the sound, graphics, and system functions of the engine.

It is possible to build a game completely from scratch, using only those header files. However, the SDK delivers a comprehensive set of classes and methods that, in its entirety, already constitutes a complete game. Starting from this point, the developer can now modify, remove or add custom parts to this framework. The advantage is rapid prototyping, as long as the result does not differ much from the original kind of game.

However, with every additional change that is necessary to get away from the original game towards the final product, it gets more and more difficult to implement the changes. Some of these difficulties are described further down in this section.

3.1.6 *Client-Server Architecture*

Games and VEs for multiple users are mostly constructed using a client/server architecture [Val10d]. The basic principle of client and server communication of a game based on the Source Engine is shown in Figure 9.

The server is mainly responsible for running the simulation, updating the position, orientation, and speed of animated and physically simulated objects. In regular intervals, e.g., every 33 ms (=30 Hz), it receives compressed command packets from the clients, carrying information about mouse movement, keyboard input, and other events that the users on the clients have triggered. These command packets are unpacked, checked, and their effect is taken into consideration for the simulation: avatars move, objects are picked up or released, sounds are played, etc. After each simulation step, the new state of all objects and avatars is sent to the clients which can in turn update the changed state of the world on the screen.

During runtime, each simulated object in the VE exists in two versions: One version, the “server entity”, is managed on the server, and is actively simulated. The second version, the “client entity”, exists on each client and is kept in sync with the server version by network variables [Val10c].

These variables automatically take care of maintaining a synchronous state between the server and all clients. As soon as a variable value changes, its value is marked for transmission on the next update data packet from the server to the clients. To conserve bandwidth, the values are being compressed and only sent when they have changed. This mechanism is important to enable fluid gameplay on low-bandwidth connections, e.g., dial-up.

3.1.7 *Prediction*

The fact that clients have to wait for a data packet from the server to show the updated world has a major drawback: Users would experience a noticeable delay to their actions, especially on slow network connections.

To avoid this delay and to provide a fast and responsive game, the client predicts the response of the server

and uses this prediction for an immediate response to user input. When the client later receives the real server response, it corrects the prediction, if necessary.

For the prediction, the client needs to have the same rules and simulation routines as the server. In the SDK, this is implemented by a major duplication of code for the server and client entity representations. However, instead of physical file duplication, shared code is contained in shared source files (e.g., `physics_main_shared.cpp`) that are included in both, client and server projects.

3.1.8 Stripping Down the Engine

The next big step, after understanding the engine, was to strip the project code of unnecessary classes and entities, e.g., weapons and the player health indicator. This step proved very difficult due to numerous interdependencies within the code. Weapon related code especially, was very deeply integrated into basic classes. Removal of one class file would break several other classes. It required a lot of re-compilation passes and uncommenting of large code sections until the code would compile again.

3.1.9 Changing the Interaction

One major change in the original SDK deathmatch game style was the primary interaction type. After we had removed all weapons, we wanted to assign the left mouse button click to grabbing and releasing of physical objects, and to triggering of interactions with objects, e.g., buttons or patients.

This seemingly simple change required a lot of reworking in the code to create access methods to the objects that the user interacts with, to enable users to take objects from each other, and to log all of those interaction events.

On a visual level, we wanted the avatars to grab an object with the right hand as soon as the user would pick it up. This can be implemented with inverse kinematics (IK): When the target position of the hand is given, IK calculates the position of the animating bones of the arm so that the attached hand reaches that position exactly.

The Source Engine is capable of IK, as can be seen in *Half-Life 2 – Episode 2*, where a certain tripod character always touches uneven ground with all three feet. However, the Source SDK website states that in multiplayer games, IK is not activated due to difficulties and performance reasons on the server [Val10a].

Our queries in the developer forums resulted in a confirmation that the engine is capable of IK, but nobody was able to give an answer on how to do it.

For this reason, grabbed objects "float" in front of the avatar while they are carried around. However, this flaw

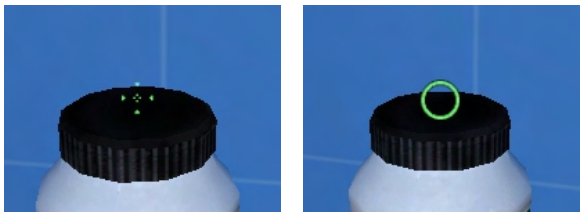


Figure 4: Different styles for the viewpoint indicator

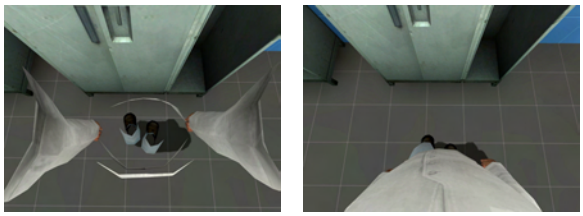


Figure 5: Creating body awareness for the avatar

Figure 6: Examples of a room with the original Source SDK textures (left) and the custom textures for the user studies

in realism did not distract the participants of the user studies. Some of them even made fun of the strange appearance, mentioning "Jedi-powers".

3.1.10 Changing the User Interface

Together with the change of the interaction style, we redesigned parts of the user interface. In the original SDK, the centre of the screen is marked by a crosshair, indicating the point where the weapon would be fired at.

With the removal of any weapon related code, the crosshair turned into a viewpoint indicator. After some experiments with different indicator styles, We chose a segmented circle that turns green as soon as an interactive object is in focus, and closes when a physical object is grabbed and held (see Figure 4). Such a circle has an improved visibility over, e.g., a simple point. It is also less associated with weapons than, e.g., a crosshair.

The original weapon crosshair was simply painted at the centre of the screen. With the inclusion of head tracking however, we also had to consider a position offset caused by the avatar head rotation and translation.

3.1.11 Body Awareness

In the original SDK, the user cannot see the avatar's own body when looking down, as shown in the left image of Figure 5.

To create body awareness, we had to change several aspects:

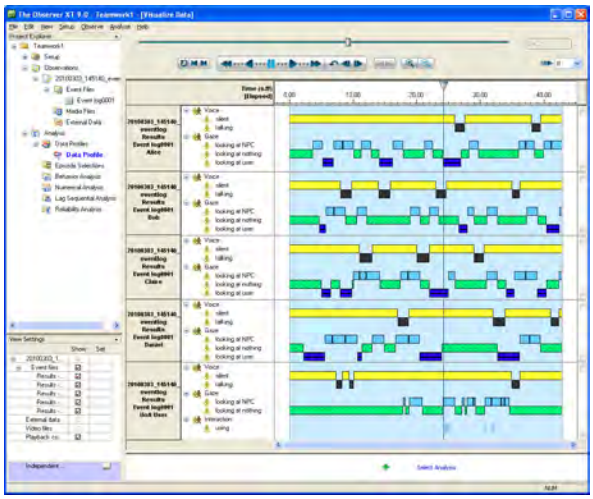


Figure 7: *Observer XT* visualising interactions, movements, and talk patterns of a teamwork simulation

1. The body model has to be drawn, even when the game is in first-person view.

2. The camera viewpoint has to be synchronised with any animation of the body model, e.g., walking, standing idle. To achieve this, the camera position is constantly updated with the position of the eyeballs of the avatar model.

3. When looking up or down, the vertical head rotation cannot simply be translated into a camera rotation, because in that case the user would be able to see the inside of the head or the body (see left screenshot in Figure 5). We added a forwards translation to the camera that is slightly increased when the user looks up or down. Together with correct settings for the near and far plane of the camera frustum, this creates a realistic body awareness without literally having "insight" into the avatar model.

We had planned to use IK to visualise the head movement caused by head tracking. Physical, translational head movement of the user would then have resulted in identical translational upper body and head movement of the avatar. As a result, an avatar would lean forward or sideways in sync with the user who is controlling it. However, we were not able to implement this feature due to the insufficient documentation of the IK features of the engine.

3.1.12 Textures

The original textures of the SDK are designed for creating games that are set in a post-war era. These textures are, in general, worn down and dull, creating a depressive feeling in all maps created with them.

We replaced some of the wall, floor, and ceiling textures with synthetic textures that look like clean tiles.

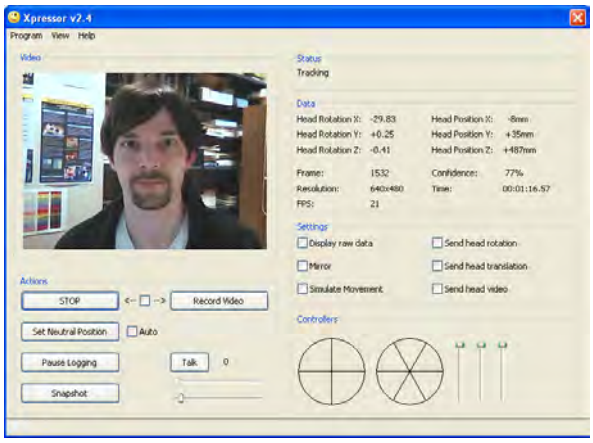


Figure 8: Screenshot of the user interface of *Xpressor*

The regular style of the tile textures creates a very organised, sterile look. The realism of the rooms created with these textures could be increased further by using photos of real rooms. However, this was not a priority for our research, but it is an indicator of the complexity of designing and creating realistic environments.

3.1.13 Data Logging

We also implemented a data logging module that records user head movement, user interactions, and gaze targets and duration. The generated logfiles enable us to analyse individual and teamwork scenarios for statistical evaluations. An additional benefit, especially for teamwork assessment, is the ability of the logfiles to be imported into external assessment tools, like the behavioural analysis tool *Observer XT* shown in Figure 7 [Nol10]. This import eliminates the need for human assessors to observe a teamwork recording again to create a list of actions and behaviours. All this information is already present in the VE engine during the simulation and can therefore be directly exported into the logfile.

3.2 Xpressor

Xpressor is the program that we developed for encapsulating the head tracking library *faceAPI*. The program communicates bidirectionally with the VE engine, using two local user datagram protocol (UDP) connections.

The communication with the VE engine occurs through a plug-in, as shown in Figure 1. The Source SDK has certain settings and abstraction layers that prevent the direct use of networking functions and several other operating system related functions. However, it is possible to load plug-in DLLs and to exchange data with them. We therefore created a simple *Xpressor* plug-in that is loaded in the beginning, accepts the UDP connection, and relays the following data into the VE engine:

- translational and rotational tracking data,

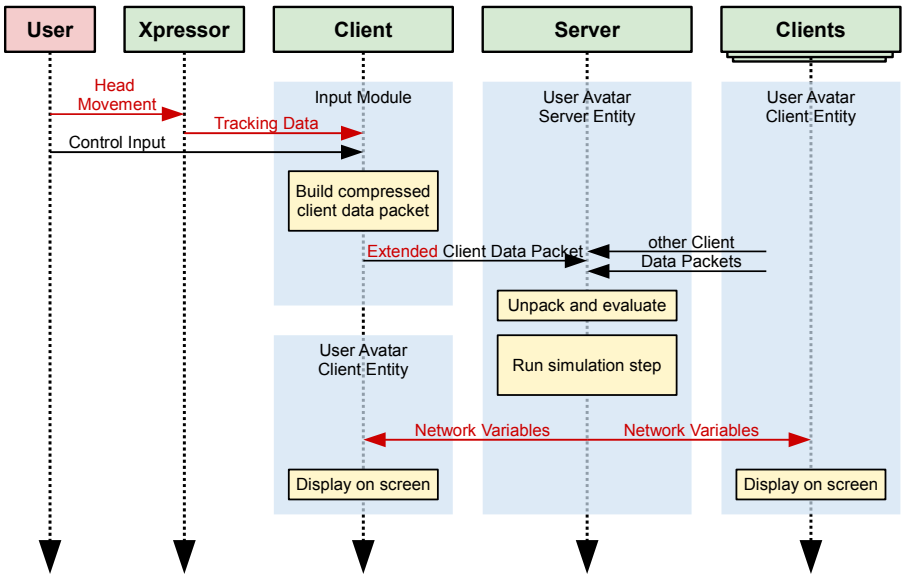


Figure 9: Data exchange between *Xpressor*, the VE clients, and the server

- a low resolution video stream,
- information regarding whether the user is speaking or not, and
- values to control the facial expression of the avatar.

The video stream is helpful for the user e.g., to adjust his or her position at the beginning of a simulation. To conserve bandwidth, the video is resized to 100×60 pixel, converted to 4 bit greyscale, and transmitted with 10 fps via a separate UDP connection.

The program also monitors the signal strength of the connected microphone, signalising the VE engine via a flag whether the user is speaking or not. The state of this flag is determined by a simple signal energy threshold algorithm.

Xpressor is written in C++, using the Microsoft Foundation Classes (MFC) for the graphical user interface (GUI) (see Figure 8). For the control of the facial expression, we developed a custom circular controller interface, visualising six expression types as circle segments and the strength of the expression by the distance of the controller position from the centre of the circle.

While sitting in front of the screen, the user inadvertently shifts his or her neutral head position relative to the camera. As a result, any concept relying on an absolute position will reflect that drift in a slowly changing view of the VE. Similar to the recommendations from Sko and Gardner for games using HCP, we have implemented a configurable automatic slow adjustment of the neutral position towards the average of the measured position over several seconds [SG09]. This adjustment accommodates for the gradual change of the neutral position and rotation of the user's head. To avoid an unwanted compensation when the user is at the extreme ends of the tracking range, e.g., when looking at an object from the side, the adjustment is reduced towards the outer regions of the tracking volume.

3.3 Data Model

The data model is a description of how to pack the values from head tracking and future facial expression recognition into a data structure that can be easily extended, but at the same time also easily compressed and transmitted.

Figure 9 visualises the extension in the data flow between the VE clients and server. Because of the fast local UDP connection between *Xpressor* and the client, the data is transferred uncompressed. Between the clients and the server however, bandwidth can be limited, therefore the parameters are compressed.

4 RESULTS

The modification of the Source Engine into a virtual environment for medical teamwork training with webcam support for HCP and head gestures was a non-trivial process due to the complexity and insufficient documentation of the engine, but allowed for rapid prototyping of early design stages.

All software outside of the VE engine, e.g., *Xpressor*, was kept modular, as well as most of the code we created to add functionality to the VE engine. This enabled us in the early stages of our experiments to easily exchange our own head tracking module by *faceAPI*.

However, features or modifications that required deep changes within the original code had to be kept close to the coding style of the SDK itself, resulting in suboptimal program code. The latter problem might be of a different magnitude when using different game engines, e.g., Unity 3D [Uni11] that provide a more structured codebase to program against. The problems with

the complexity of the code of the Source SDK were increased by insufficient documentation. A lot of development time was spent on deciphering the code or consulting the forums and developer websites for examples to compensate for the lack of documentation. To avoid this problem, it is important to put more emphasis on the quality of the documentation and the code of a game engine when engines are considered for selection.

Content for our VE was created using the free 3D editor *Blender* [Ble11] and the tools provided by the Source Engine, e.g., the map editor *Hammer* and the character animation tool *Faceposer*. Most time during content creation was spent on figuring out ways how to simulate a specific effect or physical behaviour with the engine which is optimized for fast action gameplay, not for precise simulations. On several occasions, we had to compromise between realism and the ability of the engine to simulate a specific feature. One example is the bleeding that occurs during the surgical procedure we designed for the multi-user study. The Source Engine does not provide physically correct fluid simulation. Instead, we created a particle effect that resembles a little fountain.

We measured the "success" of the design and implementation of our VE indirectly by the user studies we conducted for our overall goal: to show improvements of usability, realism, and effectiveness of VE-based training scenarios by including camera-based non-verbal communication support and intuitive HCP-based view control.

Overall, the VE proved to be stable and intuitive to use for the participants, regardless if they were experienced in playing computer games or not. Our studies comparing manual view control against HCP showed that HCP is an intuitive and efficient way of controlling the view, especially for inexperienced users [MWW10].

For highest user comfort, it is important that the delay between physical head movement and virtual camera movement is as short as possible. Our framework was able to deliver a relatively short response time of about 100 ms. However, this delay lead to participants repeatedly overshooting their view target. We suspect that the delay is a sum of several smaller delays in each processing stage of the data flow, therefore requiring several different optimisation steps for an improvement.

For our latest user study, we created a surgical teamwork training scenario and alternated between HCP and avatar control being enabled or disabled to investigate the effect of tracking-based avatar head movement on non-verbal communication within a VE. The results showed an increase in perceived realism of the communication within the environment [MWW11]. An effect on teamwork training effectiveness was not proven, but might have been masked by the experiment design. A clarification is subject to future research.

5 CONCLUSION

In summary, the Source Engine is suitable for rapidly developing a teamwork training VE, as long as the changes required to the original SDK code are not too major. The more functionality that is necessary for specific features of the desired VE, the more complex the coding task becomes. At a certain point, it would be infeasible to use this engine and alternative game engines would have to be considered.

However, the Source Engine proved stable and flexible enough for our medical teamwork training scenario with additional support for HCP and camera-controlled avatar head gestures. The user studies we have conducted show that these extensions are well received, and improve the usability and the perceived realism of the simulation. In addition, the digital recording of the interactions and behaviours within the VE is a valuable support for automated (e.g., with tools like *Observer XT*) as well as "manual" assessment of teamwork performance.

6 REFERENCES

[Ble11] Blender Foundation. Blender, 2011. `http://www.blender.org`.

[BLWM08] Marian Bartlett, Gwen Littlewort, Tingfan Wu, and Javier Movellan. Computer Expression Recognition Toolbox. In *Demo: 8th Int'l IEEE Conference on Automatic Face and Gesture Recognition*, 2008.

[DPH+09] Douglas Danforth, Mike Procter, Robert Heller, Richard Chen, and Mary Johnson. Development of Virtual Patient Simulations for Medical Education. *Journal of Virtual Worlds Research*, 2(2):3–11, August 2009.

[Lin10] Linden Research, Inc. Second Life, 2010. `http://secondlife.com`.

[MRL+06] Brian MacNamee, Pauline Rooney, Patrick Lindstrom, Andrew Ritchie, Frances Boylan, and Greg Burke. Serious Gordon: Using Serious Games To Teach Food Safety in the Kitchen. In *Proceedings of the 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games (CGAMES06)*, November 2006.

[MSL+09] Paul R. Messinger, Eleni Stroulia, Kelly Lyons, Michael Bone, Run H. Niu, Kristen Smirnov, and Stephen Perelgut. Virtual Worlds – Past, Present, and Future: New Directions in Social Computing. *Decision Support Systems*, 47(3):204–228, June 2009.

[MWW07] Stefan Marks, John Windsor, and Burkhard Wünsche. Evaluation of Game Engines for Simulated Surgical Training. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 273–280, New York, NY, USA, December 2007. ACM.

[MWW10] Stefan Marks, John Windsor, and Burkhard Wünsche. Evaluation of the Effectiveness of Head Tracking for View and Avatar Control in Virtual Environments. *25th International Conference Image and Vision Computing New Zealand (IVCNZ) 2010*, November 2010.

[MWW11] Stefan Marks, John Windsor, and Burkhard Wünsche. Head Tracking Based Avatar Control for Virtual Environment Teamwork Training. In *Proceedings of GRAPP 2011*, 2011.

[Nol10] Noldus Information Technology. Observer XT, 2010. `http://www.noldus.com/human-behavior-research/products/the-observer-xt`.

[QCM10] Rossana B. Queiroz, Marcelo Cohen, and Soraia R. Musse. An extensible framework for interactive facial animation with facial expressions, lip synchronization and eye behavior. *Computers in Entertainment (CIE) - SPECIAL ISSUE: Games*, 7:58:1–58:20, January 2010.

[RLD06] Andrew Ritchie, Patrick Lindstrom, and Bryan Duggan. Using the Source Engine for Serious Games. In *Proceedings of the 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games (CGAMES06)*, November 2006.

[SG09] Torben Sko and Henry J. Gardner. Human-Computer Interaction — INTERACT 2009. In Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, Raquel Oliveira Prates, and Marco Winckler, editors, *Lecture Notes in Computer Science*, volume 5726/2009 of *Lecture Notes in Computer Science*, chapter Head Tracking in First-Person Games: Interaction Using a Web-Camera, pages 342–355. Springer Berlin / Heidelberg, August 2009.

[SJB07] Tarja Susi, Mikael Johannesson, and Per Backlund. Serious Games -– An Overview. Technical report, School of Humanities and Informatics, University of Skövde, Sweden, February 2007.

[ST09] Shamus P. Smith and David Trenholme. Rapid prototyping a virtual fire drill environment using computer game technology. *Fire Safety Journal*, 44(4):559–569, May 2009.

[The11] The Apache Software Foundation. Apache Subversion, 2011. `http://subversion.apache.org`.

[TSHW08] Jeffrey Taekman, Noa Segall, Eugene Hobbs, and Melanie Wright. 3DiTeams — Healthcare Team Training in a Virtual Environment. *Simulation in Healthcare: The Journal of the Society for Simulation in Healthcare*, 3(5):112, 2008.

[Uni11] Unity Technologies. UNITY: Unity 3 Engine, 2011. `http://unity3d.com/unity/engine`.

[Val07] Valve Corporation. Source Engine, 2007. `http://source.valvesoftware.com`.

[Val10a] Valve Developer Community. IK Chain, 2010. `http://developer.valvesoftware.com/wiki/$ikchain`.

[Val10b] Valve Developer Community. My First Mod, 2010. `http://developer.valvesoftware.com/wiki/First_Mod`.

[Val10c] Valve Developer Community. Networking Entities, 2010. `http://developer.valvesoftware.com/wiki/Networking_Entities`.

[Val10d] Valve Developer Community. Source Multiplayer Networking, 2010. `http://developer.valvesoftware.com/wiki/Net_graph`.

[Val10e] Valve Developer Community. Using Subversion for Source Control with the Source SDK, 2010. `http://developer.valvesoftware.com/wiki/Using_Subversion_for_Source_Control_with_the_Source_SDK`.

A morphing approach for kidney dynamic modeling From 3D reconstruction to motion simulation

Valentin Leonardi
LSIS, UMR CNRS 7296
Campus de Luminy
13288 Marseille cedex 9,
France
valentin.leonardi@univ-amu.fr

Jean-Luc Mari
LSIS, UMR CNRS 7296
Campus de Luminy
13288 Marseille cedex 9,
France
jean-luc.mari@univ-amu.fr

Vincent Vidal
L2PTV, EA 4264
CERIMED
13385 Marseille cedex 5,
France
vincent.vidal@ap-hm.fr

Marc Daniel
LSIS, UMR CNRS 7296
Campus de Luminy
13288 Marseille cedex 9,
France
marc.daniel@univ-amu.fr

ABSTRACT

Motion simulation of an organ can be useful in some cases like organ study, surgery aid or tumor destruction. When using a non-invasive way of tumor destruction through transcutaneous transmition of waves, it is primordial to keep the wave beam focused on the tumor. When the tumor is not in movement, such a task is trivial. But when the tumor is located in a moving organ like the kidney, motion simulation is necessary. We present here an original method to obtain the kidney motion simulation: this is done using a mesh morphing (we consider the kidney has already been segmented and reconstructed for three different phases of the respiratory cycle). Such an approach allows a smooth transition between the different kidney models, resulting in a motion simulation. Thus, the method is purely geometric and does not need any kind of markers or tracking device. It gives directly a full 3D simulation and models are animated in real time. Finally, our approach is automatic and fast, so that it can easily be used in a medical environment.

Keywords: Geometrical modeling, organ motion simulation, kidney modeling, mesh morphing

1 INTRODUCTION

Tumors can be treated by low-invasive approaches. The goal is to minimize interactions between the surrounding environment and the patient in order to limit the consequences of surgery (incision treatment, convalescence) and their possible complications (nosocomial infections). Kidney tumors can be treated by radiofrequency. Radiofrequency is a low-invasive, non-surgical percutaneous heat treatment. The principle is to locate the tumor through CT scan, and insert a radiofrequency electrode in its center. An electric current is then delivered, in order to destroy the tumor. However, there is a chance of cancerous cell displacement when removing the electrode.

The KiTT project (for **Ki**dney **T**umor **T**racking, of which we take part) is fully involved in the low-invasive protocol. Its goal is to create a *totally non-invasive* new approach by transmitting radiofrequency waves, in a transcutaneous way until tumor eradication. The main difficulty is to keep the wave beam continuously focused on the tumor while the kidney is deformed and moves because of the respiratory cycle. A kidney (and a tumor) *tracking* is therefore necessary. Before this organ tracking stage, we need to obtain a solid 3D model of it. This work is described in our previous paper [LMVD11].

What we propose here is a new method which has two goals: the first one is the motion and deformation visualization of an organ (the kidney in this case) under the influence of natural breathing. The second goal results directly from the first one and is the tracking of a part of this organ: its tumor. The originality of this method is that it uses a fully geometric approach: mesh morphing. Thus it is fast and only needs three models corresponding to three breathing phases: inhale phase (when volume of air in lungs is maximal), exhale phase (when volume of air in lungs is minimal) and the middle phase between the two previous ones, which we refer as the mid-cycle phase. Moreover, the results obtained are fully geometric; the output is an animated 3D model. Thereby general motion and all deformations can be studied at once where some methods only offer the possibility of a 2D visualization. Finally, as the tumor is also animated, it is possible to know its position at any time.

Section 2 of this article deals with the previous work in organ tracking and mesh morphing. Section 3 introduces the general process of our method: a brief recall of the kidney reconstruction is done, then the algorithm used for mesh morphing is described in detail.

In section 4, we present the obtained results, their performances and we discuss them. Finally in section 5, we present the limits of our method, the possibilities to overcome them and the perspectives of future work.

2 RELATED WORK

2.1 Organ tracking

Organ tracking methods are either based on mathematical models which represent the respiratory cycle as a periodical function, or on empirical algorithms which predict future movements by observation and analysis of previous ones.

The most intuitive way to track an organ is to put a **marker** which is highly detectable by a classic medical imaging acquisition near this organ [NUG+08, NPSA07, OTW+05, SGB+00, SSK+00]. This formalism is also used in all-in-one robotic radiosurgery systems such as the Cyberknife [MCG+03]. This kind of method requires a surgical intervention which is not suitable for our problematic.

The following approaches assume the kidney has been segmented and reconstructed previously for two or more phases of the respiratory cycle. Most of the time, only two models are needed, but three [SBMG06] or even more [RMOZ01] are sometimes necessary. These extra acquisitions can be used to improve the precision of the organ deformation. In other cases, it is not an extra acquisition of the organ that is needed, but other kind of data essential to the method. Hostettler *et al.* [HNS+08] use the diaphragm movement in order to reflect it on the abdomen organs. In [SBMG06], air, tissues and lungs have to be segmented for three acquisitions in order to get an organ tracking.

Deformation fields are used to understand the motion of an organ. This field computes the deformations necessary to apply on a given source model M_s to deform it into a given target model M_t. The deformation field can be computed using several methods like Maximum Likelihood / Exceptation-Maximisation [RMK+05], least squares [SBMG06] or approaches based on Normalized Mutual Information [RSH+99]. Deformations can also be applied on a mesh through a deformable superquadratic in order to get the movement of an organ [BCA96].

Registration methods are also a good way to have an organ tracking. Nicolau *et al.* [NPSA07] use two acquisitions: on the first one, markers are used in order to get the position of the organ of interest. Then a second acquisition is done without these markers. By analyzing the difference of position of the spine for both acquisitions, the registration is performed using the minimization of the Extended Projective Point Criterion. In [RSH+99] two operations are done to compute the registration: affine transformation is used for global movements while Free Form Deformation is used for local movements. Two registration algorithms based on optical flow are implemented and accelerated using GPU programming in [NdSE+08] in order to perform an image-guided radiotherapy.

2.2 Principle of morphing

Mesh morphing is a method used to transform progressively a source model M_s into a target model M_t. The most usual method to perform a mesh morphing is to find a common vertex/edge/face network for both models in order to compute a metamesh M_m which contains the topology of M_s and M_t. The common network is obtained by mapping the mesh into arbitrary shapes, using different kind of mappings based on the resolution of a linear system. This approach was first used by Kent *et al.* in [KCP92], where both models are mapped onto the unit disk. In [ACOL00] Alexa *et al.* perform a mesh morphing where the interior is also considered. A 3D mesh is decomposed into a set of tetrahedrons and a 2D shape into a set of triangles. The interpolation of a tetrahedron is done using a rotation and a scale-shear with positive scaling matrices. In [GSL+98] both M_s and M_t are divided into an equal number of patches P. Each patch is then mapped so that the patch P_k of M_s is morphed into the patch P_k of M_t. This approach is close to the one used by [KSK00] since models are also divided into n arbitrary shapes. These shapes are mapped onto a polygon, where vertices of the border of a shape lie on the border of the polygon. The position of the remaining vertices is then computed by considering the shape as a spring-mass system at rest, the border vertices being the the fixed masses. In [LDSS99] M_s and M_t are simplified into M'_s and M'_t using the MAPS method [LSS+98]. Vertices for which vertex-vertex correspondences is already known are kept. M_s and M_t are finally mapped in order to compute the correspondences. Unfortunately, the previous approaches always need either user interaction (which can be very time consuming for some methods) or vertices correspondence between M_s and M_t prior to the mesh morphing. In both case, our constraints do not allow to spend a lot of time on cutting up a mesh manually or making vertex-vertex correspondences.

It is possible to fully automate a mesh morphing algorithm by using an automatic mesh cutting up. Indeed, such a process allows to separate a model into at least two different parts which can then be mapped. Several works can be found, although it is a difficult problem: some methods are based on the use of a single patch [KSK97], where others are related to graph

theory problem and aim at balancing the size of patches [EDD+95, KK99]. However, our models are close to each other, which does not justify such an advanced approach. Another way to have a complete automatic mesh morphing is to map models onto the unit sphere [KCP92]. Indeed, there is no need to divide the models anymore since they are homeomorphic to a sphere. On the other hand, the model has to be star-shaped, which is not the case of a kidney. Alexa [ACOL00] introduces a variant for sphere mapping: as for barycentric mapping, each vertex is placed at the centroid of its neighbors. Finally, a new approach for mesh morphing is done by Lee *et al.* in [YHM07]. The principle is to compute a constraints field C in order to deform M_s into M_t. C is then interpolated in order to determine a new constraints field C' for an intermediate model M_i between M_s and M_t.

3 GENERAL PROCESS

3.1 Overview

A general view of our entire workflow is presented on Figure 1. It shows how to get the kidney motion visualization from 3 sets of images. These sets result from three CT-scan or MRI acquisitions. First, the kidney and tumor are segmented for the three sets of images. Then, the organ is reconstructed in order to have three 3D models (called M_1, M_2 and M_3), each one corresponding to a precise breathing phase. Mesh morphing between M_1 and M_2 and between M_2 and M_3 is computed. Our mesh morphing approach is based on an automatic mesh cutting up, unit disk mapping and metamesh creation. Since models are relatively close to each other, they can be divided into only two patches. Moreover, the frontier between the two patches always has the same orientation on the kidney, which allows this step to be entirely automatic. Once the two patches are defined for M_s and M_t, a unit disk mapping is performed in order to overlay the two mappings of a corresponding patch of both models. We cannot use a sphere mapping here as kidney models are not star-shaped. Detections of mapped edge intersections and mapped vertices positions allow to create a metamesh which comprises the topology of both models. As an initial and a final position are known for all metamesh vertices, the metamesh is animated by linear interpolation. The alternation between metameshes coming from M_1 and M_2 and from M_2 and M_3 gives a full kidney animation from inhale to exhale phase, *i.e.* the whole respiratory cycle. The only part of our method done in real time and during the whole tumor destruction process is the metamesh vertices interpolation. Everything before this step is done once and for all and takes less than 2 minutes (from kidney segmentation to metamesh creation).

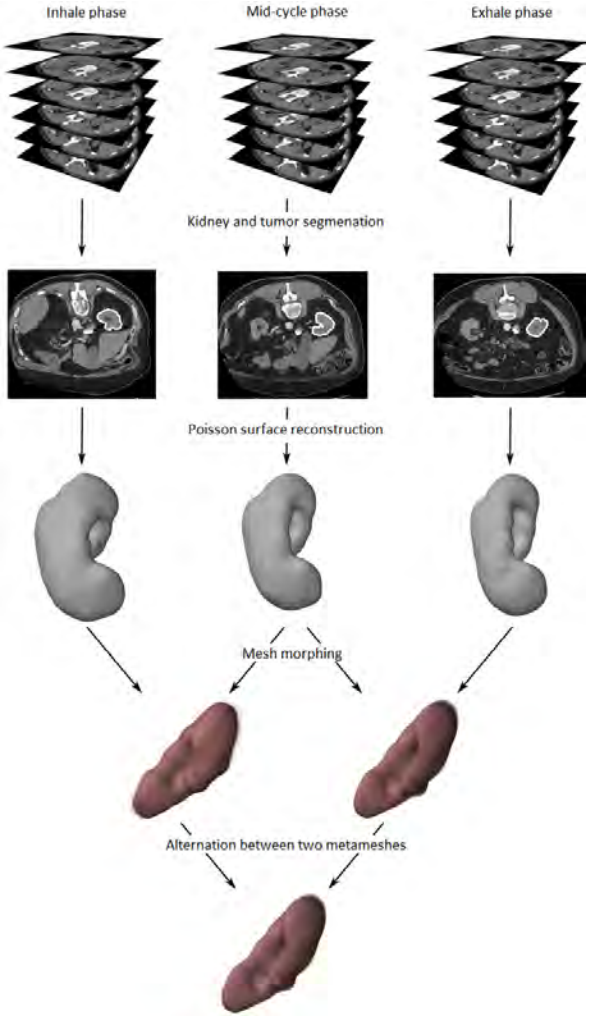


Figure 1: *Overview of our entire workflow: three sets of images resulting from a medical imaging acquisition for the inhale, mid-cycle and exhale phase is done (first line). The kidney and the tumor are segmented for every images of these three acquisitions (second line). The Poisson surface reconstruction is then applied to the point cloud extracted from the segmentation of each three different phases. We call the resulting models M_1, M_2 and M_3 (third line). Mesh morphing is computed between M_1 and M_2 and between M_2 and M_3. The results are two metameshes which allow a smooth transition between M_1 to M_2 and M_2 to M_3 (fourth line). By alternating the two metameshes, a full and smooth transition from M_1 to M_3 is possible, resulting in the kidney motion visualization (fifth line).*

3.2 Kidney reconstruction

A full description of the kidney reconstruction is detailed in [LMVD11]. The method used to get the kidney model is divided into two stages: the first one is the kidney segmentation from which a point cloud is extracted. The second stage consists in reconstructing

this point cloud in order to obtain a model.

A region growing approach is used in order to segment the kidney. Despite some methods exist to define automatically the seed needed for initialization, our approach uses a minor user-interaction and needs a single mouse click to define it. However, this is done only for one image (since the whole kidney is present in at least 60 slices). The region segmented in an image I_{k-1} is used to get the seed for the next image I_k: the weighted barycenter of the points defining the contour in I_{k-1} defines the seed for I_k. Results of this method are shown on Figure 2

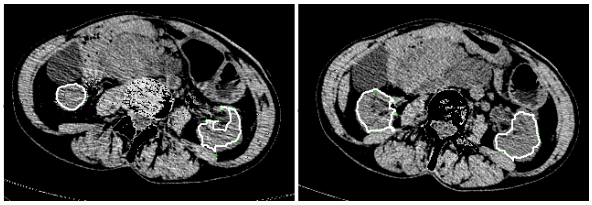


Figure 2: *Final results using our kidney segmentation approach for left and right kidney on two different slices.*

The point cloud extracted from the segmentation stage is reconstructed using the Poisson Surface Reconstruction [KBH06]. The principle of this algorithm is to define an indicator function χ peculiar to a model M, which is 0 for every point outside the model and 1 inside. Deducing χ directly from the oriented point cloud is the major problem in this case. The solution is to use the gradient of χ since the point cloud can be considered as samples of $\vec{\nabla}\chi$ (see Figure 3). Indeed, $\vec{\nabla}\chi$ is a vector field that is 0 almost everywhere except near the surface. A vector field $\vec{V}$ which is an approximation of $\vec{\nabla}\chi$ is found using the original normals. χ must now be deduced from $\vec{V}$, *i.e.* $\vec{\nabla}\chi = \vec{V}$. This is done by applying the divergence operator to express it as a Poisson equation: $\Delta\chi \equiv \nabla \cdot \vec{\nabla}\chi = \nabla \cdot \vec{V}$. The resolution of this equation is a well known problem (especially in physics) but will not be discussed here. The final reconstruction is then obtained from the extraction of an appropriated isosurface (see Figure 4).

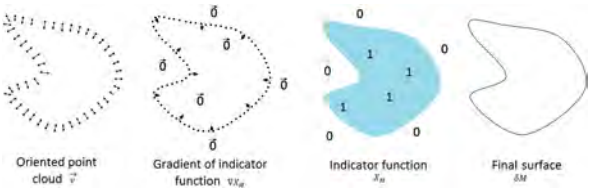


Figure 3: *Overview of the Poisson surface reconstruction.*

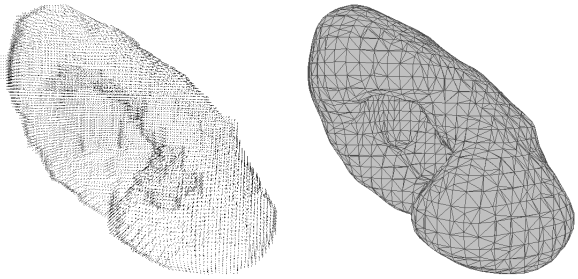


Figure 4: *Final result of a kidney point cloud (left) and its reconstruction using the Poisson surface reconstruction (right).*

3.3 Morphing

The morphing stage must have very basic user- interactions. The two models to morph are close to each other since they both come from the same kidney. Thus, the mesh morphing method uses an automatic mesh cutting up in two patches, unit disk mapping and metamesh creation. We cannot map onto a sphere as kidney models are not star-shaped. All these steps are described in detail hereunder. For the rest of this paper, we will use the following symbols: M represents a given model, M_s is the source model and M_t the target model, as used in the previous sections. C is the connectivity between vertices, edges and faces of M. $V = \{v_1, v_2, v_3, ..., v_n\}$ is the position in $\mathbb{R}^3$ of vertices. Edges are represented as a pair of vertices $\{i, j\}$ and faces as a triplet of vertices $\{i, j, k\}$. Finally, $N(i)$ is the set of adjacent vertices to vertex $\{i\}$, *i.e.* $N(i) = \{\{j\} | \{i, j\} \in C\}$.

Mesh cutting up: obtaining the *tearing path*

The first stage of the mesh dissection consists in computing its principal axis. This can be done by considering only the vertices and using Principal Component Analysis (PCA). Moreover, the PCA gives the 3 principal vectors of the mesh; the firsts two and the barycenter of the mesh define the *principal plane*. Thus, the next stage consists in computing the intersections between edges of M and the principal plane, defining what we call the *intersected edges*. In the same way, the vertices $\{i, j\}$ of an intersected edge are called *intersected vertices*. This set of the intersected edges is the first stage of the final tearing path (see Figure 5).

The tearing path must be a unique loop of edges in C, *i.e.* $\{\{i_1, i_2\}, \{i_2, i_3\}, ..., \{i_{n-1}, i_n\}, \{i_n, i_1\} \,|\, \{i_k, i_m\} \in C \; \forall (k, m) \in [1; n]$; this set of edges is a subset of C and is called c. Thus, two successive intersected edges must share a same vertex. The purpose of the first post-process of the intersected edges is to remove dead-end edges from c. Such edge has one of its vertices which is not shared with any other intersected edge, *i.e.* $\{\{i, j\} | \forall l \in N(j) \{j, l\} \notin c\}$. To detect such edges, we

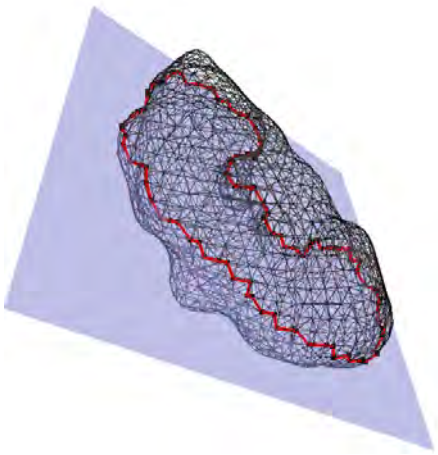


Figure 5: *Intersection between the kidney model and its principal plane (in blue). The resulting tearing path is displayed in red.*

first compute the partial adjacency list of each vertex in c. This list is the set of adjacent vertices $\{j\}$ in c to a vertex $\{i\}$, *i.e.* $\{\{j\}|\{i,j\} \in c\}$. A dead-end edge is then simply detected when at least one of its vertices has only one neighbor, *i.e.* its partial adjacency list length is 1 (see Figure 6 - b). The second post-process consists in removing *local loops*: the tearing path must be a unique succession of edges and each vertex must be shared by two and only two edges. Thanks to the partial adjacency list, vertices from which the tearing path separates are easily detected: such vertices have, at least, 3 neighbors. Thus, local loops are removed as follow. Starting from a 2-adjacency vertex we choose arbitrarily one of its neighbors and so on, until a 3-adjacency vertex is reached. During this step, each vertex is skimmed only once so that it appears at most once in the final tearing path. An arbitrary neighbor of the current 3-adjacency vertex is still chosen, but every other edges containing the current vertex is suppressed from c. As such a process creates new dead-end edges, every edge of each 2-adjacency neighbor is recursively suppressed until the neighbor is a 3-adjacency vertex (see Figure 6 - c,d,e). As the current 3-adjacency vertex becomes a 2-adjacency vertex, the whole process is repeated until we fall back on the first vertex.

Mapping mesh onto the unit disk

Once the tearing path has been computed, vertices are tagged in three different way. We call them *tag* 0, 1 and 2. Tagging the mesh allows to define the two parts of it which will be mapped later. Vertices defining the tearing path are tagged as 0. A unique arbitrary neighbor of a vertex tagged as 0 is tagged as 1. We recursively tag all its neighbors, so that a whole part of the mesh is tagged as 1. The other part is tagged as 2. Both

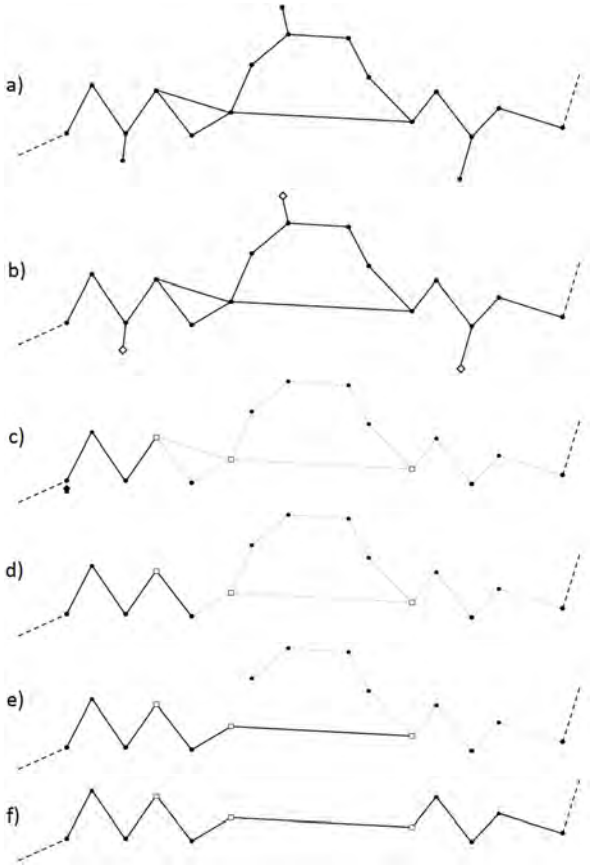


Figure 6: *Whole example of the post-process of a tearing path. Although this example cannot exist in a real situation, it presents all the cases needed to understand how the full post-process works. From top to bottom: (a) Original tearing path - (b) 1-adjacency vertex detection (diamond) and dead-end edges suppression - (c) 3 (or more)-adjacency vertex detection (square). Starting from the pointed vertex, an arbitrary neighbor is chosen. - (d) For a 3-adjacency vertex, we still choose an arbitrary neighbor, but every other edge is suppressed. - (e) To avoid apparition of new dead-end edges when edges are suppressed, recursive suppression of every edges from 2-adjacency neighbor is done. - (f) The final tearing path obtained after the post-process.*

meshes are then rotated so that their principal planes are aligned with xz-plane. This way, it is possible to check if parts tagged the same in the two models have the same y orientation. If not, tags 1 and 2 of one model are swapped. This step is essential since the part of M_s tagged as 1 (resp. 2) will be morphed into the part of M_t tagged as 1 (resp. 2) (see Figure 7).

Now that every vertex is tagged, they can be mapped onto the unit disk. Although any kind of mapping is applicable, we choose the discrete harmonic mapping [Pol00] since it preserves as much as possible the topo-

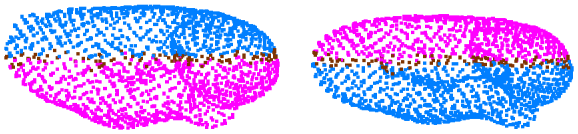


Figure 7: *Example of two models for which a same tag has a different y orientation. Vertices in red are tagged as 0, vertices in cyan tagged as 1 and vertices in magenta tagged as 2.*

logy of faces of both models. The most straightforward step of this mapping is for the intersected vertices. They are fixed on the unit circle in a way that arc length between each pair of successive vertices is proportional to the original length of edge in mesh. For vertices tagged as 1 or 2, discrete harmonic mapping (as well as other mapping) amounts to solving a linear system described as follow. Two distinct mappings are done, one for each tag. Let V_i be the vertices to map with $0 \leq i < n$ index of vertices tagged as 1 (resp. 2) and $n \leq i < N$ index of vertices tagged as 0. Then, the linear system to solve is:

$$(I-\Lambda)\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \end{pmatrix} = \begin{pmatrix} \sum_{i=n}^{N-1} \lambda_{0,i} v_i \\ \sum_{i=n}^{N-1} \lambda_{1,i} v_i \\ \sum_{i=n}^{N-1} \lambda_{2,i} v_i \\ \vdots \\ \sum_{i=n}^{N-1} \lambda_{n-1,i} v_i \end{pmatrix}$$

where $\Lambda = \{\lambda_{i,j}\}$ and $\lambda_{i,j}$ is a coefficient depending on the mapping used. Here, for discrete harmonic mappings, we have:

$$\lambda_{i,j} = \begin{cases} \frac{cot\alpha_{i,j}+cot\beta i,j}{\sum_{j\in N(i)}(cot\alpha_{i,j}+cot\beta_{i,j})} & \text{if} \quad \{i,j\} \in C \\ 0 & \text{if} \quad \{i,j\} \notin C \end{cases}$$

with $\alpha_{i,j} = \angle(i,k_0,j)$ and $\beta_{i,j} = \angle(i,k_1,j)$. Edge $\{i,j\}$ is adjacent to two and only two faces since M is a triangular mesh. k_0 and k_1 are the two vertices that define these faces. We call M'_{sN} M'_{tN} the mapping of M_s and M_t for tag N. Similarly we call $\{i'\}$ a mapped vertex. Although such notation should not exist since only the position of vertices (v_i) changed during the mapping, this notation will make further expressions more straightforward.

Metamesh creation and animation: computing intersections and barycentric coordinates

The next step is to overlay M'_{sN} M'_{tN} for both tags in order to compute the metamesh. The first stage is to detect intersections between mapped edges. When two edges $\{i',j'\} \in C$ for M'_s and $\{k',l'\} \in C$ for M'_t cross, a new vertex is created. Two valid definitions of this intersection point are $v'_i + \alpha\overrightarrow{v'_i v'_j}$ and $v'_k + \beta\overrightarrow{v'_k v'_l}$. Coefficient α and β are saved along with the new vertex. These coefficients will be necessary for intermediate models as they are sufficient to compute the coordinates of the vertex, even when v_i, v_j, v_k and v_l are interpolated. These kind of vertex is called an *intersection vertex*. Once an intersection vertex is created, appropriate edges and faces are created along with it in order to build the topology of the metamesh M_m. These new edges and faces will allow M_m to combine topology of both M_s and M_t and to have a continuous interpolation between the two models (see Figure 8).

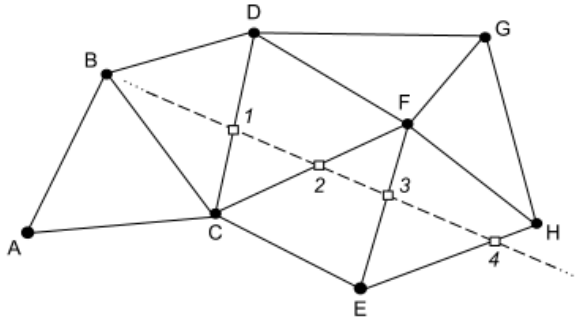


Figure 8: *Example of intersections between mapped edges of M_s (solid line) and M_t (dotted line). Intersection points 1, 2, 3 and 4 are created, as well as appropriate edges (C1, 1D, C2, 2F, ...) and faces (C12, F23, ...).*

The second stage of the metamesh creation is the computation of barycentric coordinates (BC) for every vertices of M_s and M_t. To do that, we first want to know on which mapped face $\{i',j',k'\}$ of M'_t (resp. of M'_s) a mapped vertex v'_m of M'_s (resp. M'_t) lies on. The BC are a unique triplet u,v,w such that $v'_m = uv'_i + vv'_j + wv'_k$. The face where v'_m lies on and its BC are saved. This kind of vertex is called a *mesh vertex* (see Figure 9).

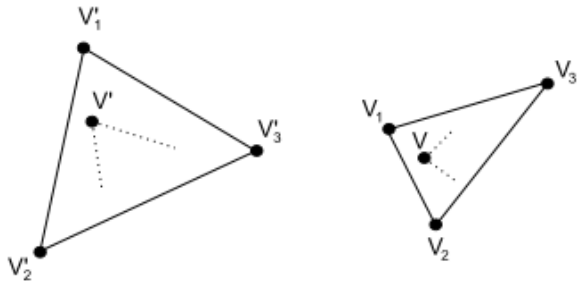


Figure 9: *Mapped vertex of M'_t, v' lies on face $\{v'_1,v'_2,v'_3\}$ of M'_s. Its BC are computed so that $v' = 0.8v'_1 + 0.7v'_2 + 0.2v'_3$. Position of v on face $\{v_1,v_2,v_3\}$ of M_s is then known thanks to these coordinates: $v = 0.8v_1 + 0.7v_2 + 0.2v_3$.*

Thus, the metamesh is completely built and composed of a set of intersection vertices and mesh vertices.

Intermediate models can now be easily obtained by interpolating positions of vertices. The interpolation is possible since we know, for each one of them, an initial and a final position as following: for a mesh vertex coming from M_s, the initial position is its position in M_s. The final position is known by the combination of its BC and the face of M_t it lies on. Inversely, for a mesh vertex coming from M_t, the initial position is known using its BC and the face of M_s it lies on. The final position is its natural position in M_t. For an intersection vertex, the initial position is known thanks to its α coefficient and the edge of M_s it lies on. The final position is computed using its β coefficient and the edge of M_t it lies on.

3.4 Tracking the tumor

The tumor tracking is the second goal of our method. It is important to know where it is located to adjust the wave beam accordingly. From this point of view, there are two main differences between the tumor and the kidney. The first one is the tumor is not deformed by the respiratory cycle, it only moves along with the kidney. The second one is the tumor is similar to an ellipsoid. In the segmentation step, the tumor is segmented separately from the kidney and in a such way that the center of the tumor is known. An other mesh morphing to obtain the tumor movements (*i.e.* its tracking) would be inappropriate since its shape remains the same from one breathing phase to another. Moreover, it would cost useless computational time. A more convenient way to do that is to interpolate the position of the tumor since we have the coordinates of its center for the inhale, exhale and mid-cycle phases. We can use a quadratic Bézier curve interpolation, which gives the tumor position for intermediate phases. In real conditions, the patient will be anesthetized and on respirator, allowing a full control on his breathing, *i.e.* the phase of his respiratory cycle is known at any time. Therefore, it is really easy to synchronize the metamesh and the tumor interpolation along with the patient's breathing. Thus, the 3D coordinates of the tumor are known at any time and correspond to its real position, resulting in the tumor tracking.

4 RESULTS

Our method has been tested on a set of three kidney models M_1, M_2 and M_3 obtained as described in section 3.2. Theses models correspond respectively to the inhale phase, mid-cycle phase and exhale phase. Two mesh morphing were performed: the first between M_1 and M_2 and the second between M_2 and M_3. Figures 11, 12 present several intermediate models obtained while performing the morphing from M_1 to M_2 to M_3. As results are not very explicit with frozen models, an animated version can be seen at the following URL: http://www.youtube.com/watch?v=dhPqLp2X8NQ.

General movement and deformation of the kidney are respected. The natural rotation of the principal axis of the organ is present here, as well as its enlargement. On the other hand, local deformations are not totally satisfying, especially for the tumor. The one on the morphed kidney is absorbed into a part of the kidney and reappear from a different part, right next to its original location. The natural deformation would have been a smooth displacement between theses two locations, almost like a translation. This is due to the morphing method itself. Although these false deformations are not really noticeable, they become obvious when tumor is displayed: it sticks out of the kidney model (Figures 10). Another drawback of our method is the cutting of the mesh by a plane. In order to have a correct morphing, the tearing path obtained from the intersection of the mesh and this plane must be composed of one connected component. Such a criterion is not always guaranteed. From our experiment and analysis, computing intersection between a kidney model and its principal plane will result in a one connect component tearing path (this is due to the shape of the organ itself), but that would not be necessarily the case for another organ or some kind of arbitrary models.

Figure 10: *Highlighting local deformation problem. Intermediate model with tumor (blue ellipsoid) presents local inaccuracy, especially for the tumor region (encircled).*

As the three models have more or less the same number of vertices, edges and faces, computation times for one morphing are equivalent for the other.s The models we have are composed by up to 2,300 vertices, 6,900 edges and 4,600 faces. A morphing is computed in *40s*, each step being repeated twice, one for each tag (data was processed on a laptop with an Intel Core i7 processor and 4 Go of RAM). Although that does not allow to compute mesh morphing in real time, this execution time is acceptable for our medical environment, where interventions used for our non-invasive tumor destruction (**Hi**gh **F**ocused **U**ltrasound) are very long (up to 3 hours). Moreover, the whole computation time is almost needed only for the metamesh creation, which is done only once. Its animation can be done in real time as it is simply an interpolation between an initial and final position of its vertices as seen in section 3.3.

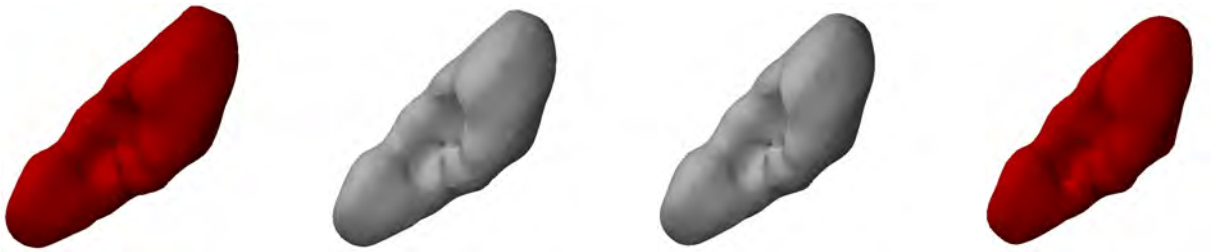


Figure 11: *Final results showing natural movements of the right kidney due to respiration. Source and target models obtained from reconstruction are displayed in red. Intermediate models are displayed in grey. Morphing from M_1 to M_2 is showed here (from left to right).*

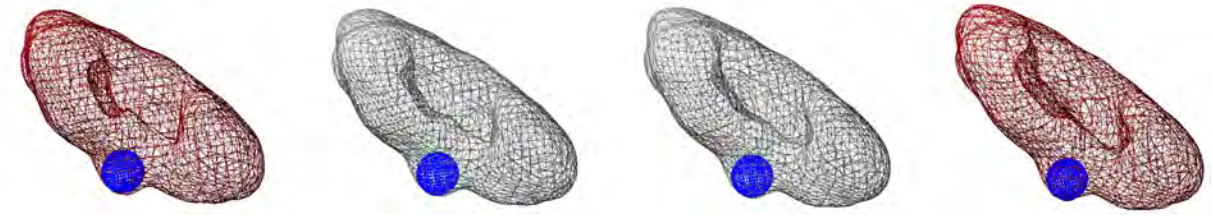


Figure 12: *Morphing between M_2 to M_3 from a different point of view (rotation of 180 degrees around vertical axis). Models are displayed in wireframe and the tumor is visible (blue ellipsoid).*

5 CONCLUSION

We have presented an original and geometric approach to obtain the natural motion simulation of the kidney under the respiratory cycle. Starting from three medical imaging acquisitions of the organ, each one for a different phase of the cycle, kidney is first segmented then reconstructed in order to create one model for each phase. Kidney is finally animated in 3D and respiratory movements are simulated through mesh morphing among the three models we previously had (from first to second model and from second to third). To do that, it is first necessary to cut the mesh, which is done automatically here. Then the two different parts of a mesh are mapped onto the unit disk. This mapping is used to compute a metamesh which comprises the topology of two successive models. Soft transition between two models, and thus the kidney animation, is finally obtained by interpolating each vertex of the metamesh between an initial and a final position. Although general deformation and movement of the kidney is well simulated, local deformations are not precise enough, especially for tumors near the surface. A way to overcome this problem would be to force regions with similar curvature to morph into each other.

ACKNOWLEDGMENT

This work is granted by the Foundation "Santé, Sport et Développement Durable", presided by Pr. Yvon Berland. The authors would like to thank everyone involved in the KiTT project: Christian Coulange for his precious help, Marc André, Frédéric Cohen and Philippe Souteyrand for their wise advices and for providing CT scan data, and Pierre-Henri Rolland for his support.

REFERENCES

[ACOL00] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. *Proceedings of Computer Graphics and Interactive Techniques*, 2000.

[BCA96] Eric Bardinet, Laurent Cohen, and Nicholas Ayache. Tracking and motion analysis of the left ventricle with deformable superquadrics. *Medical Image Analysis*, 1(2):129 – 149, 1996.

[EDD+95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Micheal Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Proceedings of SIGGRAPH*, pages 173 – 182, 1995.

[GSL+98] A. Gregory, A. State, M.C Lin, D. Manocha, and M.A. Livingston. Feature-based surface decomposition for correspondence and morphing between polyhedra. *Proceedings of Computer Animation*, pages 64 – 71, 1998.

[HNS+08] Alexandre Hostettler, Stéphane Nicolau, Luc Soler, Yves Rémond, and Jacques Marescaux. A real-time predictive simulation of abdominal organ positions induced by free breathing. *International Symposium on Biomedical Simulation*, pages 89 – 97, 2008.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. *Eurographics Symposium on Geometry Processing*, 2006.

[KCP92] James Kent, Wayne Carlson, and Richard Parent. Shape transformation for polyhedral objects. *Computer Graphics*, 26(2), July 1992.

[KK99] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999.

[KSK97] T. Kanai, H. Suzuki, and F. Kimura. 3d geometric metamorphosis based on harmonic map. *Proceedings of The Fifth Pacific Conference on Computer Graphics and Applications*, 1997.

[KSK00] Takashi Kanai, Hiromasa Suzuki, and Fumihiko Kimura. Metamorphosis of arbitrary triangular meshes. *Proceedings of Computer Graphics and Application*, 20(2), March 2000.

[LDSS99] Aaron Lee, David Dobkin, Win Sweldens, and Peter Schroder. Multiresolution mesh morphing. *Proceedings of Computer Graphics and Interactive Techniques*, 1999.

[LMVD11] Valentin Leonardi, Jean-Luc Mari, Vincent Vidal, and Marc Daniel. Reconstruction 3d du volume rénal à partir d'acquisitions scanner volumiques. *Journée du Groupe de Travail en Modélisation Géométrique, GTMG*, pages 83 – 92, March 2011.

[LSS+98] Aaron Lee, Win Sweldens, Peter Schroder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH*, pages 95 – 104, July 1998.

[MCG+03] Martin Murphy, Steven Chang, Iris Gibbs, Quynh-Thu Le, Jenny Hai, Daniel Kim, David Martin, and John Adler. Patterns of patient movement during frameless image-guided radiosurgery. *International Journal of Radiation Oncology Biology Physics*, 55(5):1400 – 1408, 2003.

[NdSE+08] Karsten Ostergaard Noe, Baudouin Denis de Senneville, Ulrik Vindelev Elstrom, Kari Tanderup, and Thomas Sangild Sorensen. Acceleration and validation of optical flow based deformable registration for image-guided radiotherapy. *Acta Oncology*, 47(7):1286 – 1293, 2008.

[NPSA07] Stéphane Nicolau, Xavier Pennec, Luc Soler, and Nicholas Ayache. Clinical evaluation of a respiratory gated guidance system for liver punctures. *Medical Image Computing and Computer-Assisted Intervention*, pages 77 – 85, 2007.

[NUG+08] Masahiko Nakamoto, Osamu Ukimura, Inderbir Gill, Arul Mahadevan, Tsuneharu Miki, Makoto Hashizume, and Yoshinobu Sato. Realtime organ tracking for endoscopic augmented reality visualization using miniature wireless magnetic tracker. *Medical Imaging and Augmented Reality*, pages 359 – 366, 2008.

[OTW+05] B. Olbricha, J. Trau, S. Wiesner, A. Wicherta, H. Feussner, and N. Navab. Respiratory motion analysis: Towards gated augmentation of the liver. *Computer Assisted Radiology and Surgery*, 1281:248 – 253, 2005.

[Pol00] K. Polthier. Conjugate harmonic maps and minimal surfaces. Technical report, Technische University of Berlin, 2000.

[RMK+05] Mauricio Reyes, Grégoire Malandain, Pierre Malick Koulibaly, Miguel Gonzalez Ballester, and Jacques Darcourt. Respiratory motion correction in emission tomography image reconstruction. *Medical Image Computing and Computer-Assisted Intervention*, pages 396 – 376, 2005.

[RMOZ01] Torsten Rohlfing, Calvin Maurer, Walter O'Dell, and Jianhui Zhong. Modeling liver motion and deformation during the respiratory cycle using intensity-based free-form registration of gated MR images. *Medical Imaging 2001: Visualization, Image-Guided Procedures, and Display*, pages 337 – 348, February 2001.

[RSH+99] Daniel Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Nonrigid registration using free-form deformations: Application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8), August 1999.

[SBMG06] David Sarrut, Vlad Boldea, Serge Miguet, and Chantal Ginestet. Simulation of four-dimensional CT images from deformable registration between inhale and exhale breath-hold CT scans. *Medical Physics*, 33(3), March 2006.

[SGB+00] Achim Schweikard, Greg Glosser, Mohan Bodduluri, Martin Murphy, and John Adler. Robotic motion compensation for respiratory movement during radiosurgery. *Journal of Computer-Aided Surgery*, 2000.

[SSK+00] H. Shirato, S. Shimizu, K. Kitamura, T. Nishioka, K. Kagei, S. Hashimoto, H. Aoyama, T. Kunieda, N. Shinohara, H. Dosaka-Akita, and K. Miyasaka. Four dimensional treatment planning anf fluoroscopic real-time tumor tracking radiotherapy for moving tumor. *International Journal of Radiation Oncology Biology Physics*, 48:435 – 442, September 2000.

[YHM07] Han-Bing Yan, Shi-Min Hu, and Ralph Martin. 3d morphing using strain field interpolation. *Computer Science and Technology*, 1, 2007.

Rendering of Translucent Objects, Verification and Validation of Algorithms

Victor A. Debelov
Institute of Comp. Math. & Math. Geophysics
SB RAS
Prospect Lavrentieva, 6
630090, Novosibirsk, Russia
debelov@oapmg.sscc.ru

Dmitry S. Kozlov
Novosibirsk State University
Pirogova str., 2
630090, Novosibirsk, Russia
kozlov@oapmg.sscc.ru

ABSTRACT

An approach to verification and validation of algorithms that render transparent objects (media) is described. Rendering of transparent *optically isotropic* objects has been studied extensively. However, the papers devoted to optically anisotropic objects are few in number. The main goal of the present paper is to suggest a collaboration in creating and supporting an open database of tests. To prepare a real scene with a crystal specimen, to photograph it, and to describe the corresponding virtual scene is a complex problem. Obviously this is an almost impossible task for many devolopers of rendering algorithms. Well-known examples of translucent media are crystals. They are convenient for testing purposes as they have permanent solid shapes. Although the crystals are rendered by a recursive ray tracing algorithm, the tests considered in the paper can be applied to other algorithms.

Keywords

Anisotropic media, crystal, birefringence, pleochroism, optical dispersion, ray tracing, polarized light, rendering algorithm, test repository, verification, validation.

1. INTRODUCTION

The difference between isotropic and anisotropic media is explained in [Hay06]: "There are two types of optical crystals, isotropic and anisotropic crystal. The isotropic crystals have the same refractive index for all directions. The anisotropic crystal has a different refractive index in a different direction, and has two different values for the same direction. However, there are one or two particular directions where these two refractive indices have the same value. These particular directions are called optic axes and the crystal having one optic axis is called uniaxial or monoaxial crystal, while the one having two optic axes is called biaxial crystal. Since the anisotropic crystal has two refractive indices, there are two refracted rays in crystal for one incident ray and so-called double refraction or birefringence occurs".

In the literature on computer graphics there are not many papers on crystal rendering devoted to rendering by polarized light and the optical phenomena (the major properties are optical dispersion, birefringence, and pleochroism). A comprehensive review of early works can be found in [Guy04] and [Wei08]. The latter paper is devoted to rendering of uniaxial monocrystals. The paper [Deb12] describes an algorithm to render isotropic and uniaxial crystalline aggregates. The paper [Lat12] describes also computations of refracted rays in biaxial crystals.

The above papers provide with different 3D scenes. It would be useful to combine the test sets. This would allow third parties to provide improvements of the algorithms.

A good example is the site [Mat], which represent a repository of test data for use in comparative studies of the algorithms of numerical linear algebra. Any new numerical algorithm has an opportunity to be examined for accuracy and steadiness and compared with the other algorithms.

A similar role in global illumination is played by the Cornell Box [Cor]. The Cornell Box has become a de facto standard. It contains an exhaustive test scene definition. Later this test was extended to semitransparent surfaces; see [Far05]. In [Smi00] are proposed a number of tests devoted to global illumination algorithms which are available on the Internet. The description of the tests is a paper text. It would be more useful to separate the papers devoted

to a description of simulation and those with test specifications.

In paper [Deb10] an approach is proposed to create a database for verification and validation of algorithms that render transparent optically anisotropic objects. As an appendix to the paper an Internet resource is made with exact specifications of test scenes' geometry and illumination. At the present time the resource is under reconstruction to adjust the ideas proposed in the above paper.

We recall that the purpose of the present paper is to create a common test database for the algorithms that render optically anisotropic translucent objects. No analysis and criticism of the algorithms are given. We describe our process of verification and validation in section 2. Section 3 is devoted to a survey of tests used by developers. Section 4 describes a real scene, techniques, and tricks used to recover the geometry and other parameters. A typical virtual scene is considered in Section 5. Section 6 describes some additional parameters of the rendering process to determine uniquely the calculated image. In section 7 additional useful tests are described. In section 8 conclusive remarks on our test repository are given.

2. PROCESS OF VALIDATION

After a local model of light interaction with transparent optically anisotropic crystalline media [Deb12] was developed we decided to create a rendering algorithm to verify and validate the model. Our algorithm is based on recursive ray tracing. One criterion of validation is to reach the maximal coincidence between a photograph of a real scene and a simulated image of a virtual scene similarly to an approach used in the Cornell Box project. In other words, the purpose is to compute a visually plausible image corresponding to a photograph of a real scene.

The algorithm is implemented in the following steps:

1. Selection of a test mineral specimen. It must be a transparent monocrystal with strong birefringence.
2. Cutting the specimen to obtain a simple geometric shape. In our case it is a hexahedron.
3. Recovery of the geometry.
4. Determination of scene illumination.
5. Creating a real test scene of crystal and light sources.
6. Selection of a photo camera.
7. Obtaining a photograph of the real scene.
8. Creating a virtual scene, i.e., a computer model of the test scene.
9. Selection of the computer model of the virtual camera. In our case it is a well-known pin-hole camera.
10. Single view calibration of the pin-hole camera from the photograph of the real scene.
11. Spectral Rendering. It is necessary to have the spectral characteristics of all objects of the real test scene or their approximate values. While rendering a spectral image representation is obtained. We call it a SRGB image.
12. Tone reproduction: transformation of the SRGB image to a RGB image.
13. Visual comparison of the photograph and the synthesized image.
14. Pixel by pixel comparison of the photograph and the synthesized image. The image resolution must be equal to the photograph resolution.

The purpose of this comparison is to test the assumptions and operation of the algorithm and determine the field of its application.

Below we will consider the steps in detail.

3. RELATED WORK

In [Guy04] several crystals are described without detailed specifications of the specimens geometry, camera, optical axis, etc. Although the authors claimed that models of standard gemstone cuts are readily available on the Internet they performed an additional investigation to justify geometry of an available specimen of tourmaline. Authors worked also on the acquisition of light illuminating their test scene. No data on the geometry obtained, virtual camera, and lighting (cube-map) are available in the paper.

Another paper [Wei08]: a calcite specimen was used for a real test scene. Again, the paper contains no detailed specifications of the specimen geometry, landscape grid pattern, virtual camera, optical axis, and illumination.

The paper [Lat12] describes algorithms based on a recursive numerical method in contrast to both papers mentioned above, which contain closed form formulas for exact calculations. Actually, this paper allows calculating the refracted rays for a ray incident onto a boundary of a crystal of different types: isotropic, uniaxial, and biaxial. The authors produce test images using a convex calcite plate in a virtual scene. The paper includes computed images that illustrate birefringence. Note, that exact geometry of virtual scenes and camera parameters, illumination, and image resolution are not specified in the text. Although the algorithms [Lat12] do not compute the correct colors of the resulting images, those images allow comparing with images obtained

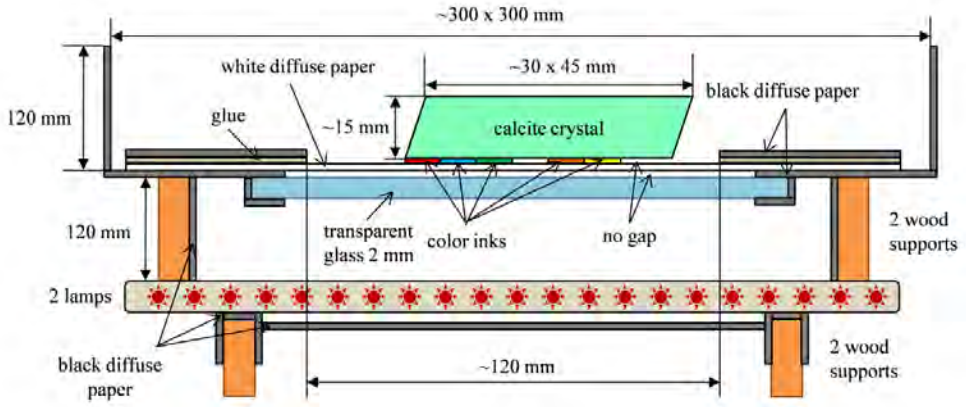


Figure 1. Scheme of a real scene.

by other algorithms in order to examine the correctness of birefringence.

The paper [Deb12] is devoted to derivation of closed form formulas to calculate a local illumination model for interaction of a light ray with the boundary of two transparent media of isotropic and uniaxial optical types. It includes also: a photograph of a calcite uniaxial crystal over a color checkered pattern, a corresponding virtual scene with the recovered geometry of the specimen, a synthesized image, and several virtual test scenes with corresponding rendered images. In spite of the fact that the paper contains detailed description of validation experiments, nevertheless, the data presented are insufficient to reproduce them thoroughly.

The paper [Deb10] has a similar purpose to create a testbed for algorithms rendering transparent isotropic and anisotropic media with polarized light. There the following groups of tests are suggested in the paper:

- Low level test based on some clearly formulated physical laws like Snell's law, Brewster's angle, etc. These tests allow us to assess, e.g., the accuracy of algorithms.
- Special virtual scenes demonstrating the physical phenomena: ortoscopic and conoscopic images, internal conical refraction, etc. [Bor80]. The authors thoroughly describe a 3D scene to render the effect of internal conical refraction. Instead of giving the exact scene specifications they describe some guidelines how to construct such scenes. Since the camera parameters and computed image are not described, it is difficult to compare the algorithms.
- Comparison of a real scene photograph with a computed image of the corresponding virtual scene. Note that the photograph of a real scene in the paper is much different from the image of the virtual one. Moreover, the real calcite specimen used is not a monocrystal but a crystalline aggregate. Besides, the camera parameters are not known.

No detailed specifications of the virtual test scenes to be rendered are available also in the papers mentioned above.

We suggest a different approach: to create a special appendix that stores exact specifications of the virtual test scenes described in a paper. It may be a personal Internet site. Note that the data presented at the site recommended in our paper [Deb10] are also incomplete and poorly documented. At the present time this site is being reconstructed.

Our main objections concern fact that the information on test scenes is often incomplete and does not allow one to reproduce the rendering of scenes and comparing of the images obtained. However, all the above-mentioned papers present numerous pictures that illustrate the approaches and/or algorithms being described. Also a limited length of papers does not allow authors to present all details which could overload a text. We expand an idea proposed in [Deb10] and suggest creating a testbed as a common Internet resource devoted to detailed information on the test data. We expect that new efficient algorithms to render anisotropic crystals will appear in the near future. This test repository, being an expandable database, can be helpful in debugging them.

4. REAL SCENE

A real scene is selected in steps 1–7 mentioned in section 2. We spent more than a year to find a crystal, cut it, take a photograph, and calibrate the virtual camera. The above repository can be helpful to save time and efforts of the developers of renderers. Otherwise the community will be limited to those who have stones, able to cut a specimen and calibrate the virtual camera, etc. The majority of researchers are programmers rather than geologists or jewelers. Therefore all photographs of proper specimens or gemstones are important. First we consider specimens that have simple geometric shapes: cube, sphere, or convex polyhedron. Jewelry shapes are more complex and require additional efforts to recover the exact geometry, see [Guy04].

It seems that recovering the geometry of transparent crystals is a difficult task. For example, calcite is too fragile, so the faceted shape of a specimen may get chipped and peeling. The KDP (a Potassium Dihydrogen Phosphate) crystals are not waterproof (expired air also), and require additional care, e.g., mask, gloves, etc. The crystals are not expensive, and have visible strong birefringence. Measuring the linear sizes of edges by standard tools may produce errors due to the possible chipped vertices of the specimen.

Often knowledge of the construction of a real scene may be useful. A scheme of a real scene and its approximate sizes are shown in Fig. 1. A specimen lies on a sheet of white paper with a printed color texture. The sheet is put on a transparent glass plate and covered by a sheet of black paper with a rectangular window. Two cylindrical luminescent lamps are placed under the plate in such a way as to provide an approximately uniform illumination of the sheet. Other light sources are absent. Specifically, a window with the printed texture and lamps determine the geometrical shape and spectrum of the light source.

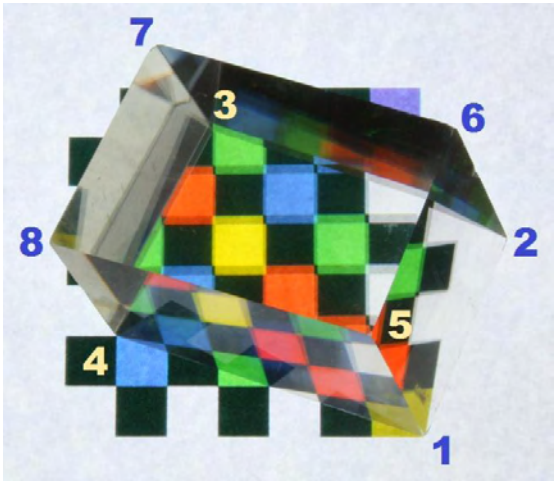


Figure 2. Photograph of a calcite specimen.

Consider a photo with enumerated vertices (Fig.2). In our experiment we took a calcite specimen (hexahedron). Using a scanner we obtained some images of the faces (like one in Fig.3) and determined the lengths of edges of all faces, see Fig.4.

A source of illumination in the scene is a rectangle with a color texture, Fig.2. There were problems of coincidence of the spectra of real and virtual textures: a) the spectra of the lamps being used are usually unknown; b) the spectra of transmittance of the paper and inks are unknown; c) the lens adds some unknown distortions to the spectrum of transmitted light; d) the sensitivity of the camera matrix to various parts of the spectrum is unknown too, etc. In our calculations we used a light source as shown in

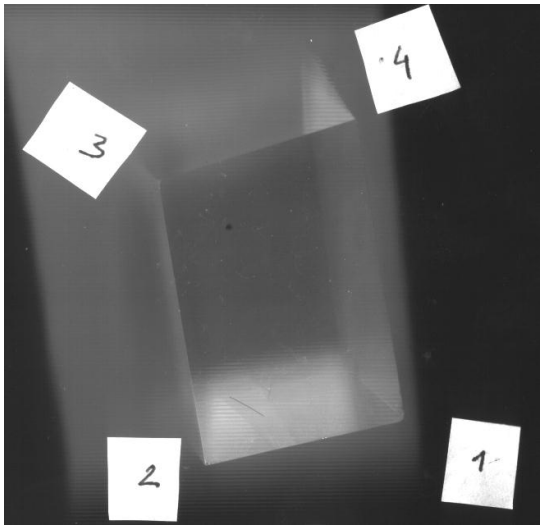


Figure 3. Scan of the face with vertices 1234. The fat black dot is the exit of the optical axis.

Fig.5. In order to decrease the difference between the textures colors in the photo and the calculated image ones, the colors of the virtual texture can be taken from a blurred (unfocused) photograph of the real texture made with the same camera parameters and exposition.

The only optical axis was determined from scans of two opposite faces, 1234 and 5678, a look through which results in the absence of doubling of the texture. On each face we selected a dot that belonged to the axis. An important point: the dots were selected manually, which can lead to small errors in the direction of the axis.

Some specifications of the photo camera being used may be useful too, for example, a camera Canon 450D with a lens EFS18-55 mm F:3.5-5.6 IS. A minimal matrix sensitivity of ISO100 was selected to decrease noise. To minimize the lens's aperture, the

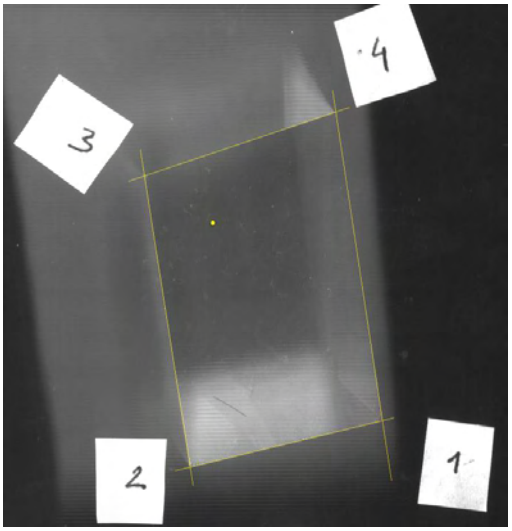


Figure 4. Scan of the face with vertices 1234. Lines approximating edges are shown.

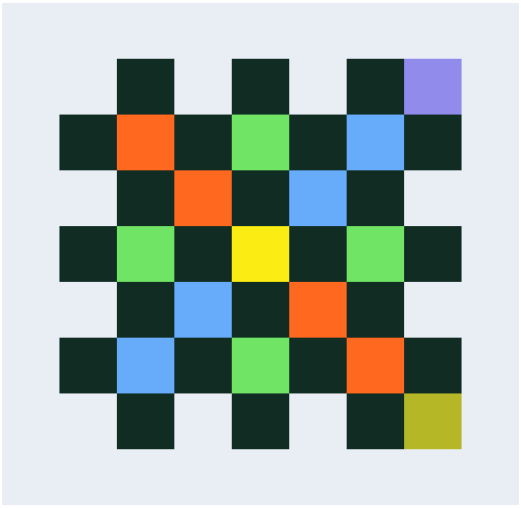


Figure 5. The texture used as a light source in the virtual scene.

diaphragm was set to a maximal value of F:36. In this case diffraction does not affect significantly the image. The exposition must be taken considerably longer than the blinking lamp period (if luminous lamps are used).

The vertices of the hexahedron are obtained as the intersection points of lines approximating the edges (Fig.6), e.g., vertex 7 in Fig.2.

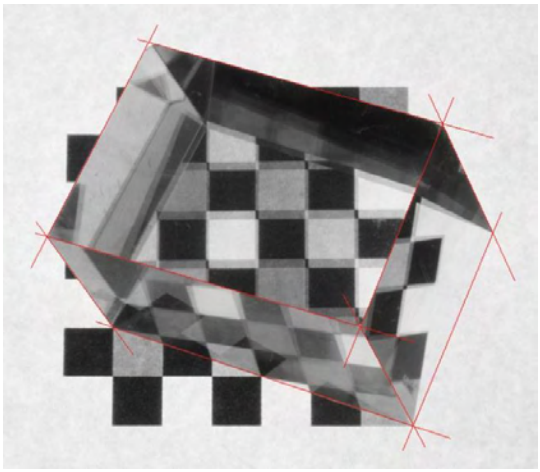


Figure 6. Gray scan of the photograph with lines approximating edges.

Thus, we have obtained some specifications of the geometry and illumination of the virtual scene with some errors at each step.

Finally, the camera parameters (camera calibration) were determined manually with the help of an interactive application. A photograph (Fig.2) and a gray scan (Fig.6) were used for calibration.

5. VIRTUAL SCENE

Our test environment is quite similar to that used in [Guy04], [Wei08], [Deb10], [Deb12]. It is not clear what illumination was used in [Lat12] as according to [Lat12, Fig.13] the test scene was illuminated by a point light source at the camera tip.

A typical virtual scene is shown in Fig.7: a specimen, a camera, and an axis aligned box around the specimen. The latter can be positioned and resized by the user. It plays the role of a cube-map which determines illumination. Arranging the box the user can put one of specimen's faces just onto the box's face. In such a way virtual scenes similar to those used in [Wei08] and [Deb12] were constructed. The six textures used allow one flexibility with assigning of illumination of the scene.

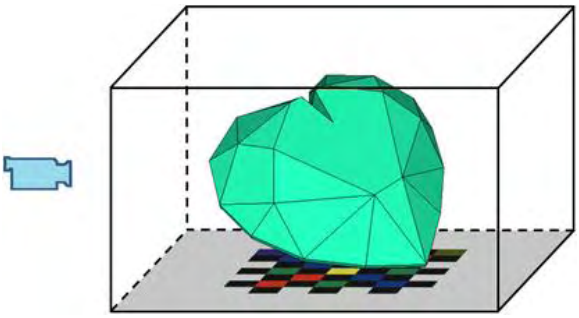


Figure 7. Virtual scene environment.

Note that all the textures are one-sided, and are invisible from outside. They illuminate unpolarized light inside the box. It is best to use a spectral representation but, in practice, only a RGB representation is available. Other *explicitly defined formats* can be applied, e.g., those in [Guy04]. Various formats can be considered as additional information, and a source of data. We believe that the spectra being used must always be provided.

Generally, the scene is filled with air or vacuum, sometimes, it may be any of transparent media: isotropic, uniaxial, and biaxial. The substance is called filler.

The following optical characteristics are specified for the scene filler and the specimen:

1) Axes. The only optical axis $\mathbf{c}$ is specified for a uniaxial medium and two axes $\mathbf{c}_1$ and $\mathbf{c}_2$ for a biaxial one. For biaxial media the directions of optical axes depend on the wavelength therefore they must be specified for each wavelength used in the image calculation, even if the directions are calculated. Obviously, isotropic media require no axis specification.

2) Main indices of refraction: one for an isotropic medium n_i, two for a uniaxial (n_o, n_e), and three for a biaxial one (n_1, n_2, n_3). The corresponding values are specified for each wavelength used in the image calculation, even if the calculation of samples is done with the Sellmeier's equation, Laurent's equation [Bor80], or in another way.

3) Absorption. In the case of a transparent colorless medium no absorption data are required. In the general case the modeling of absorption is challenging problem because of the following two facts. First, the refraction and absorption properties are not independent [Bor80]. Second, the rays propagating in absorbing anisotropic media are elliptically polarized but in transparent anisotropic media they are linearly polarized [Bor80]. These facts are usually ignored, see e.g. [Guy04]. Therefore majority of crystalline media require the determination of one, two, or three attenuation spectra. Apparently, this format satisfies the absorption data from [Guy04].

Similarly to the above remark about the format of illumination data, we believe that some parameters of a virtual scene as the spectra must be determined.

Note that a source of possible errors in the final per pixel comparison of a photo and a calculated image are the unknown physical conditions of the specimen, namely, the temperature, external forces and fields, etc. These parameters are usually taken from references where they are evaluated accordingly to certain conditions.

All the virtual scenes must be placed in the repository, even if there are no corresponding photographs of the real scenes. This may help in comparing the algorithms being used.

6. VIRTUAL TEST SCENES

A most important group consists of tests of comparison of the photograph with the computed image is as considered above.

A second group of tests consists of virtual scenes that do not require the existence of the corresponding real scenes.

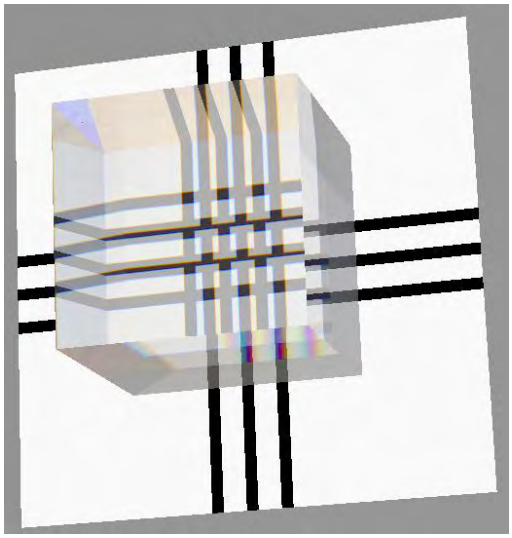


Figure 8. Computed image of a biaxial cube.

Various virtual scenes are created during the debugging process. In Fig.8, Fig.9, and Fig.10 three typical examples are shown. For the tests the repository includes:

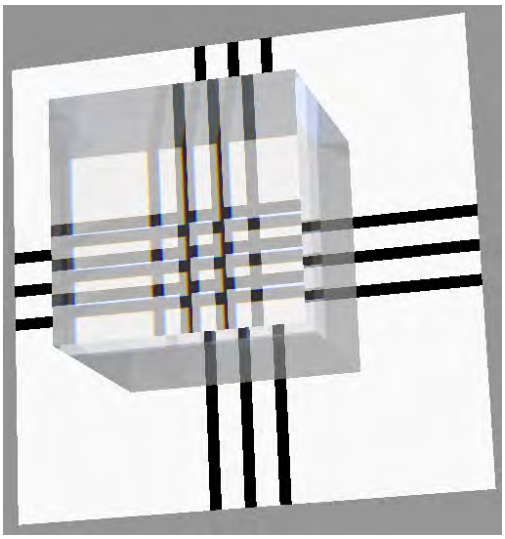


Figure 9. Computed image of a uniaxial cube.

- An image of a test scene rendered with OpenGL. It is used to comment the scene geometry and simplify the understanding.
- Coordinates of cube vertices.
- Coordinates of a square axes-aligned textured plate.
- Texture image.
- Light environment.
- Ray tracing depth in bounces.
- Pin-hole camera parameters: focus length, aperture point, view direction.
- Image plane sizes.
- Image resolution.

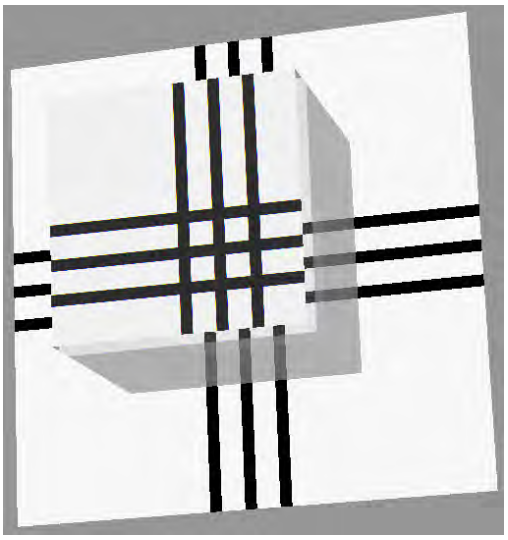


Figure 10. Computed image of an isotropic cube.

- Optical type of the scene filler: isotropic, uniaxial, biaxial. In our case it is isotropic.
- Refractive index of the filler.
- Optical type of the specimen: biaxial (Fig.8), uniaxial (Fig.9), or isotropic (Fig.10).
- Representation of the spectra: number of samples from 380nm to 780nm.
- Main refractive indices for each sample.
- Calculated image.
- Several copies of the calculated image with appropriate comments (optional).

It is now possible to calculate the image of the specified scene and provide its per pixel comparison with the image from the repository.

7. RENDERING

The repository may include several calculated images for a single virtual scene. They can be different because of the following rendering parameters:

- Image resolution.
- Number of wavelengths used (e.g., one for monochromatic light). All the optical parameters (see previous section) should be represented by spectra of equal length to represent the smooth part of the spectrum and the set of separate peaks to represent the another part.
- Depth of ray tracing. A recursive ray tracing is very time consuming, since at each bounce the ray is split into up to four generated rays. For example, if the data are: depth=20, spectra of 21 samples the rendering took several hours of calculation on a 8-processor cluster.

Figure 11. Calculated image corresponding to the photograph in Fig.2.

- Camera parameters given in the scene coordinate system. For example, we used pin-hole camera in the algorithm.

Each image is specified by its computer platform and the computational time. This will facilitate in comparing the performance of the algorithms.

A result of rendering our virtual scene is shown in Fig.11.

8. OTHER TESTS

Virtual Scenes Illustrating Some Well-Known Facts From Optics

Internal conical refraction is a phenomenon observed in biaxial crystals. The conoscopic images of anisotropic crystals are a practical means in petrography; see [Bor80] for a theoretical foundation. In [Deb10] corresponding virtual scenes are described. These tests can help in debugging and comparing the rendering algorithms.

Additional Tests

This group of tests is targeted to help in the debugging of separate program blocks. The description may be reduced to a minimum. We suggest that the authors contributing to the repository may only name a fact and refer to a proper source from the well known literature. For example, "Brewster Angle Test", see [xx, page yyy]. A better way is a detailed description of a test including the specifications of: a) the optical characteristics of two media; b) the ray incident onto the boundary between the media; c) polarization of the ray; d) the resulting rays (reflected and refracted) with their polarization states.

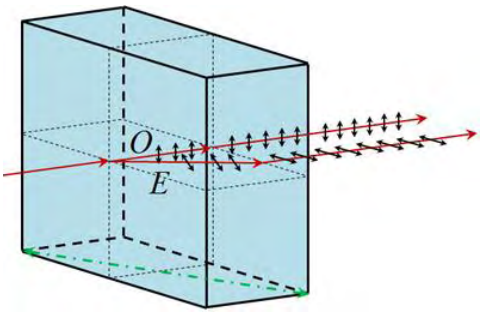


Figure 12. Birefringence test, arrows show polarization state.

In [Deb10] several examples are given: Snell's law, Brewster's angle, and a birefringence test. Consider the last one. In Fig.12 a typical scene is presented. Not only the ray directions be verified, but the polarization state of the generated rays as well. This can be useful if the repository includes several examples of optical media with incident rays and derived rays of different polarization states and intensities.

The problem of numerical stability may arise in debugging, e.g., when the directions of the ray, the optical axis, and the normal are almost the same. We believe that such situations must be put into the

repository too, especially, the problem has been solved.

9. CONCLUSIONS

We did not intend to cover every possible situation in the rendering of crystals. In this paper we have presented our approach. The repository must not have a certain predefined format. Tests of any complexity delivered by any developers are welcome. The contributing developer may deliver his/her information in any convenient format. Nevertheless, we have remarks: a) lossless image formats like BMP, PNG should be used; b) the spectra should be used wherever it is possible. In case other formats are used they should be transformed to the BMP or spectra formats explicitly. This will help in avoiding possible uncertainties.

Additional information of any kind will be very useful, for example: an OpenGL image of the recovered geometry (see Fig.7), references to the relevant papers and reports, images with comments, a photograph or a scheme of a real scene (Fig.1), and appropriate unpublished comments.

We suggest separating all tests in the repository into the following groups:

- A photograph of a real scene,
- Virtual scenes,
- Virtual scenes of optic phenomena,
- Tests of particular features,
- Other tests.

Additionally, a set of keywords should be supported; each keyword refers to the relevant tests. The set of keywords contains: real scene photographs; recovery of specimen's geometries; definitions of illumination, transparent media, absorbing media, isotropic media, uniaxial media, biaxial media, camera calibration, etc.

We do not believe that we have found a complete solution. This paper was caused by the present time situation with accessible tests.

An Internet site, [Crt], was developed initially as a support for the paper [Deb10]. At the present time it is under reconstruction. Nevertheless, the reader can find a detailed description of the tests corresponding to Fig.8-11.

The problem of creating test scenes based on real scene photographs is not an easy task. Experts on a wide variety of research have to be involved: crystallographers, specimen cutters, etc. Also a wide range of specific devices have to be used: spectrometers, etc.

We hope that this paper will help in creating a repository with the specifications described above. The inclusion of any test into our website is welcome.

10. ACKNOWLEDGMENTS

This work was supported in part by the Russian Foundation for Basic Research, grants No. 12-07-00386 and No. 12-07-00391.

11. REFERENCES

[Bor80] Born, M. and Wolf, E. Principles of optics: electromagnetic theory of propagation, interference and diffraction of light. Cambridge: Cambridge University Press, 1980.

[Cor] Cornell Box: http://www.graphics.cornell.edu/online/box

[Crt] Crystal tests: http://oapmg.sscc.ru/temp_crystal_tests/

[Deb10] Debelov, V.A., Kozlov, D.S. Verification of algorithms of photorealistic rendering of crystals. Proc. Graphicon'2010, Russia, St.Petersburg, September 20-24, 2010, pp. 238–245. (In Russian). http://www.graphicon.ru/proceedings/2010/Proceedings.pdf

[Deb12] Debelov, V.A., Kozlov, D.S. A local model of light interaction with isotropic and uniaxial transparent media. Vestnik of Novosibirsk State University, Series: Information Technologies, vol. 10, No. 1, pp. 5–23, 2012, (in Russian).

[Far05] Farnsworth, M., Erbacher, R. F. Global illumination: efficient renderer design and architecture. Proc. Intern. Conf. on Geometric Modeling, Visualization & Graphics, pp. 1691-1695, 2005.

[Hay06] Hayamitzu, Y. Analysis of internal conical refraction using ray tracing formulas for the biaxial crystal. Optical review 13, No. 4, pp.169–183, 2006.

[Guy04] Guy, S. and Soler, C. Graphics gems revisited. ACM Trans. on Graphics (Proceedings of the SIGGRAPH conference) 23, No. 3, pp.231–238, 2004.

[Lat12] Latorre, P., Seron, F. J., and Gutierrez, D. Birefringence: calculation of refracted ray paths in biaxial crystals. The Visual Computer 28, No. 4, pp. 341-356, 2012.

[Mat] Matrix Market: http://math.nist.gov/MatrixMarket/

[Smi00] B. Smits, and H. W. Jensen. Global illumination test scenes. Tech. Rep. UUCS-00-013, Computer Science Department, University of Utah, June 2000.

[Wei08] Weidlich, A. and Wilkie, A. Realistic rendering of birefringency in uniaxial crystals. ACM Transactions on Graphics 27, (1):6:1–6:12

WSCG 2012

Index