ISSN 1213-6972 *Volume 20, Number 1-3, 2012*

Journal of WSCG

An international journal of algorithms, data structures and techniques for computer graphics and visualization, surface meshing and modeling, global illumination, computer vision, image processing and pattern recognition, computational geometry, visual human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

Vaclav Skala – Union Agency

Journal of WSCG

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

Managing Editor: Vaclav Skala

Printed and Published by:
Vaclav Skala - Union Agency
Na Mazinach 9
CZ 322 00 Plzen
Czech Republic

Hardcopy: ***ISSN 1213 – 6972***
CD ROM: ***ISSN 1213 – 6980***
On-line: ***ISSN 1213 – 6964***

ISBN 978-80-86943-78-7

Journal of WSCG

Editor-in-Chief

Vaclav Skala

c/o University of West Bohemia
Centre for Computer Graphics and Visualization
Univerzitni 8
CZ 306 14 Plzen
Czech Republic

http://www.VaclavSkala.eu

Journal of WSCG URLs: http://www.wscg.eu or http://wscg.zcu.cz/jwscg

Editorial Advisory Board
MEMBERS

Journal of 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)

Journal of WSCG

Vol.20

Contents

Coupling between Meshless FEM Modeling and Rendering on GPU for Real-time Physically-based Volumetric Deformation

Muhammad Mobeen Movania
Nanyang Technological University, Singapore
mova0002@e.ntu.edu.sg

Feng Lin
Nanyang Technological University, Singapore
asflin@ntu.edu.sg

Kemao Qian
Nanyang Technological University, Singapore
mkmqian@ntu.edu.sg

Wei Ming Chiew
Nanyang Technological University, Singapore
chie0017@e.ntu.edu.sg

Hock Soon Seah
Nanyang Technological University, Singapore
ashsseah@ntu.edu.sg

ABSTRACT

For real-time rendering of physically-based volumetric deformation, a meshless finite element method (FEM) is proposed and implemented on the new-generation Graphics Processing Unit (GPU). A tightly coupled deformation and rendering pipeline is defined for seamless modeling and rendering: First, the meshless FEM model exploits the vertex shader stage and the transform feedback mechanism of the modern GPU; and secondly, the hardware-based projected tetrahedra (HAPT) algorithm is used for the volume rendering on the GPU. A remarkable feature of the new algorithm is that CPU readback is avoided in the entire deformation modeling and rendering pipeline. Convincing experimental results are presented.

Keywords

Volumetric deformation, physically based deformation, finite element method, meshless model, GPU transform feedback, volume rendering

1. INTRODUCTION

Interactive visualization of physically-based deformation has been long pursued as it plays a significant role in portraying complex interactions between deformable graphical objects. In many applications, such subtle movements are necessary, for example, surgical simulation systems in which a surgeon's training experience is directly based on the feedback he/she gets from the training system.

Prior to the advent of the Graphics Processing Unit (GPU), such interactions were only restricted to sophisticated hardware and costly workstations. Thanks to the massive processing capability of modern GPUs, such interactions can now be carried out on a consumer desktop or even a mobile device. However, even with such high processing capability, it is still difficult to simultaneously deform and visualize a volumetric dataset in realtime. Several promising volumetric deformation techniques have been proposed, but they have mostly favored a specific stage of the programmable graphics pipeline. Therefore, these approaches could not utilize the full potential of the hardware efficiently.

With new hardware releases, new and improved features have been introduced into the modern GPU. One such feature is transform feedback in which the GPU feedbacks the result from the geometry shader stage back to the vertex shader stage. While this method was usually used for dynamic tessellation and level-of-detail (LOD) rendering, we have proposed to use this mode for an efficient deformation pipeline. Since this deformation uses the vertex shader stage, we may streamline the fragment shader stage for volume rendering, forming a coupled graphics pipeline.

In Section 2, we report a comprehensive survey on deformation algorithms and GPU acceleration technologies. Then, we describe our new meshless FEM approach and the formulation of the physical model in Section 3. In Section 4, we present the techniques for coupling between the novel deformation pipeline and the GPU-based volume rendering. Experimental results and comparisons of the performance are given in Section 5. And finally, Section 6 concludes this paper.

2. PREVIOUS WORK

Up to now, physically-based deformation can be broadly classified into mesh-based and meshless methods. Mesh-based methods include finite element method (FEM), boundary element method (BEM), and mass spring system. Meshless methods include smoothed point hydrodynamics (SPH), shape matching and Lagrangian methods. We refer the reader for meshless methods to [ST99], [HF00] [BBO03], [NRBD08] and for physically-based deformation approaches in computer graphics to [NMK06].

One of the first mass spring methods for large deformation on the GPU for surgical simulators is attributed to Mosegaard et al. [MHSS04], in which Verlet integration is implemented in the fragment shader. Using the same technique, Georgii et al. [GEW05] implemented a mass spring system for soft bodies. The approach by Mosegaard et al. [MHSS04] requires transfer of positions in each iteration. Georgii et al. [GEW05] thus focused on how to minimize this transfer by exploiting the ATI Superbuffers extension. They described two approaches for implementation: an edge centric approach (ECA) and a point centric approach (PCA). A CUDA-based mass spring model has been proposed recently [ADLETG10].

All of the mass spring models and methods discussed earlier used explicit integration schemes which are only conditionally stable. For unconditional stability, implicit integration could be used as demonstrated for the GPU-based deformation by Tejada et al. [TE05].

The mass spring models are fast but inaccurate. FEM methods have been proposed for more accurate simulation and animation. The model assumes linear elasticity so the deformation model is limited to small displacements. In addition, the small strain assumption produces incorrect results unless the corotational formulation is used [MG04] which isolates the per-element rotation matrix when computing the strain.

With the increasing computational power, non-linear FEM has been explored, in which both material and geometric non-linearities are taken into consideration [ML03], and [ZWP05]. The fast numerical methods for solving FEM systems for deformable bodies are based on the multi-grid scheme. These approaches have been extended in animation [SB09] and medical applications for both the tetrahedral [GW05] [GW06] and hexahedral FEM [DGW10].

In addition to the above approaches, explicit non-linear methods have been proposed using the Lagrangian explicit dynamics [MJLW07] which are especially suitable for real-time simulations. A single stiffness matrix could be reused for the entire mesh. Especially with the introduction of the CUDA architecture, the Lagrangian explicit formulation has been applied for both the tetrahedral FEM [TCO08] as well as the hexahedral FEM [CTA08].

The problem with explicit integration is that it is only conditionally stable, that is, for convergence, the time step value has to be very small. In addition, such integration schemes may not be suitable during complex interactions as in surgery simulations and during topological changes (for example, cutting of tissues). Allard et al. [ACF11] circumvent these cons by proposing an implicitly integrated GPU-based non-linear corotational model for laparoscopic surgery simulator. They use the pre-conditioned Conjugate Gradient (CG) solver for solving the FEM. They solve the stiffness matrices directly on the mesh vertices rather than building the full stiffness assembly. Ill-conditioned elements may be generated in the case of cutting or tearing which may produce numerical instabilities.

3. THE MESHLESS FEM APPROACH

Although there have been significant achievements in deformable models, a few difficulties in the mesh-based models still exist in real-time volumetric deformation, as highlighted in the followings:

- Approximating a volumetric dataset requires a large number of finite tetrahedral elements. Numerical solution of such a large system would require a large stiffness matrix assembly. This makes the model unsuitable for real-time volumetric deformation. In addition, the corotated formulation is needed which further increases the computational burden.
- The solution of the tetrahedral FEM requires an iterative implicit solver for example Newton Raphson (Newton) or Conjugate Gradient (CG) method. These methods converge slowly. Moreover, the implicit integration solvers reduce the overall energy of the system causing the physical simulation to dampen excessively.
- Even though multi-grid schemes are fast, they have to update the deformation parameters across different grid hierarchy levels. This requires considerable computation. Moreover, the number

of grid levels required is subjective to the dataset at hand and there is no rule to follow for accurate results.

On the other hand, we have noticed that the meshless FEM approach has not been applied for volumetric deformation in the literature. Our preliminary study shows that the meshless formulation possesses a few advantages:

- It supports deformations without the need for stiffness warping (the corotated formulation).
- The solution of meshless FEM is based on a semi-implicit integration scheme which not only is stable but also converges faster as compared to the implicit integration required by the tetrahedral FEM solver. In addition, it does not introduce artificial damping.
- It does not require an iterative solver such as conjugate gradient (CG) method which is required for conventional FEM.

Therefore, in this study, we are interested in exploiting the meshless FEM approach for volumetric deformation, coupled with simultaneous GPU-based real-time visualization.

Formulation of the Physical Model

We base our deformation modeling and rendering on the continuum elasticity theory. Key parameters in the physical model are stress, strain and displacement. Strain (ε) is defined as the relative elongation of the element. Assuming an element undergoing a displacement (ΔL) having length (l), the strain may be given as:

$$\varepsilon = \frac{\Delta L}{l}$$

For a three-dimensional problem, the strain (ε) is represented as a symmetric 3×3 tensor. There are two popular choices for the strain tensor in computer graphics, the linear Cauchy strain tensor given as

$$\varepsilon_{Cauchy} = \frac{1}{2}(\nabla U + [\nabla U]^T) \quad (1)$$

and the non-linear Green strain tensor given as

$$\varepsilon_{Green} = \frac{1}{2}(\nabla U + [\nabla U]^T + [\nabla U]^T \nabla U) \quad (2)$$

In Eq. (1) and (2), the ∇U is the gradient of the displacement field U. Similar to the strain, in the three-dimensional problem, the stress tensor (σ) is also given as a 3×3 tensor. Assuming that the material under consideration is isotropic and it undergoes small deformations (geometric linearity), the stress and strain may be linearly related (material linearity) using Hooke's law, given as

$$\sigma = D\varepsilon \quad (3)$$

Since stress and strain are symmetric matrices, there are six independent elements in each of them. This reduces the isotropic elasticity matrix (D) to a 6×6 matrix as follows:

$$D = B\begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

$$B = \frac{E}{(1+\nu)(1-2\nu)}$$

where, E is the Young's modulus of the material which controls the material's resistance to stretching and ν is the Poisson's ratio which controls how much a material contracts in the direction transverse to the stretching.

In a finite element simulation, we try to estimate the amount of displacement due to the application of force. There are three forces to consider (see Fig. 1):

- Stress force (σ) which is an internal force,
- Nodal force (q) which is an external force applied to each finite element node, and
- Loading force (t) which is an external force applied to the boundary or surface of the finite element.

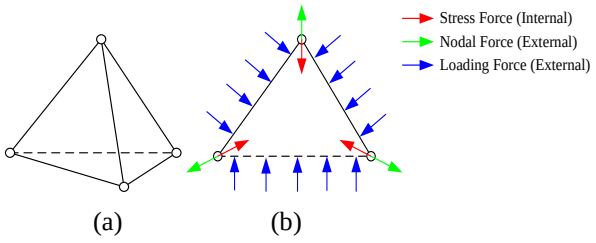


Figure 1. Different forces acting on a finite tetrahedral element (a), with (b) its cross sectional view highlighting the different internal and external forces acting on the finite element

For the finite element to be in static equilibrium, the amount of work done by the external forces must be equal to that of the internal forces, given as

$$\int_{V^e} W_\sigma dV = W_q + \int_{A^e} W_t dA \quad (4)$$

where, W_σ is the internal work done per unit volume by stress σ, W_q is the external work done by the nodal force q on the element's node and W_t is the external work done by the loading force t on the element per unit area. V^e is the volume and A^e is the area of the finite element e. W_σ is given as

$$W_\sigma = (\delta\varepsilon)^T \sigma$$

where, $\delta\varepsilon$ is the strain produced by the stress σ. Similarly, W_q is given as

$$W_q = (\delta u^e)^T q^e$$

where, δu^e is the displacement of the finite element e produced by the force q^e. W_t is given as

$$W_t = (\delta u)^T t$$

Substituting these in Eq. (4), we get

$$\int_{V^e} (\delta\varepsilon)^T \sigma dV = (\delta u^e)^T q^e + \int_{A^e} (\delta u)^T t dA \quad (5)$$

Since δu^e provides the displacement of node e at vertices only, to get the displacement at any point within the finite element, we can interpolate it with the shape function $\boldsymbol{N}$. After applying a differential operator $\boldsymbol{S}$ to the shape functions, we get the change in strain ($\delta\varepsilon$). The matrix product ($\boldsymbol{SN}$) can be replaced by B

$$\delta\varepsilon = B\delta u^e$$

Substituting ($\delta\varepsilon$) in Eq. (5), we get

$$\int_{V^e} (B\delta u^e)^T \sigma dV = (\delta u^e)^T q^e + \int_{A^e} (N\delta u^e)^T t dA \quad (6)$$

Simplifying Eq. (6), taking the constant terms out of the equation and solving integral (see Appendix) gives

$$B^T DBu^e V^e = q^e + \int_{A^e} (N^T t) dA \quad (7)$$

The left side is replaced by the element stiffness matrix ($K^e = B^T DBV^e$) and the right by the element surface force (f^e), which gives us the stiffness matrix assembly equation:

$$K^e u^e = q^e + f^e \quad (8)$$

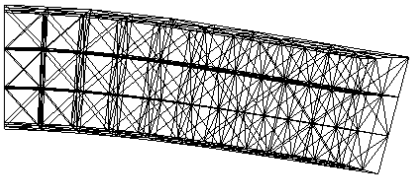
The Meshless FEM

The conventional FEM methods discretize the whole body into a set of finite elements. Calculation of the element stiffness matrix in Eq. (8) requires the volume of the body which is represented as the sum of the finite elements' volume. For instance, the widely used corotated linear FEM [MG04] has to construct the global stiffness matrix for each deformation frame, and the matrix is then solved using an iterative solver such as the conjugate gradients (CG). This makes the implementation inefficient for a large volume.

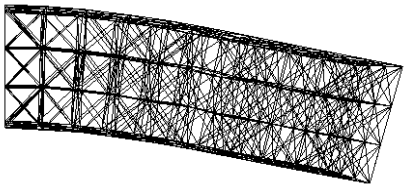
In our meshless FEM, the whole body is sampled at a finite number of points. The typical simulation quantities such as the position (x), velocity (v) and density (ρ) are all stored with the points, and the displacement field is estimated from the volume of the point and its mass distribution. The gradient of the displacement field is then estimated to obtain the Jacobian. Finally, the Jacobian is used to calculate the stresses and strains. These, in turn, allow us to obtain the internal forces. Since the meshless FEM uses the moving least square approximation, it does not require the stiffness matrix assembly, enabling a much better execution performance.

To ascertain that our proposed meshless FEM is able to produce the same deformation as that in the conventional FEM such as the corotated linear FEM, we conducted a computational experiment on a horizontal beam as shown in Fig. 2. The two results show a horizontal beam having Young's modulus of 500,000 psi and the Poisson ratio of 0.33. The dimensions of the two beams are the same. The beam in Fig. 2 (a) contains 450 tetrahedra for the corotated linear FEM whereas its equivalent one in Fig. 2 (b) contains 176 points for the meshless FEM.

While the two computations yield the same deformation under the given load, our meshless FEM has a significantly improved execution performance: 40 msecs per frame by the corotated linear FEM, compared to 1.25 msecs per frame with the meshless FEM. These timings include both the deformation as well as rendering time.



(a)



(b)

Figure 2. Comparison of deformation of a horizontal beam using (a) linear FEM and (b) meshless FEM

In a dynamic simulation, we are to solve the following system:

$$m\ddot{x} = -c\dot{x} + \sum (f_{\text{int}} + f_{ext}) \quad (9)$$

The first term on the right is the velocity damping term with c being the damping coefficient. For an infinitesimal element, the mass is approximated using density (ρ). This changes Eq. (9) to

$$\rho\ddot{x} = -c\dot{x} + \sum (f_{\text{int}} + f_{ext}) \quad (10)$$

The external forces (f_{ext}) are due to gravity, wind, collision and others. Since our system assumes geometric and material linearity, Eq. (10) becomes a linear PDE that may be solved by discretizing the domain of the input dataset using finite differences over finite elements. This system may be solved using either explicit or implicit integration schemes.

Smoothing Kernel

In the conventional FEM, the volume of the body is estimated from the volume of its constituent finite elements. Calculation of the element stiffness matrix requires the volume of the body which is usually represented as the sum of the finite elements volume. In the case of the meshless FEM, it is approximated from the point's neighborhood. For each point, its mass is distributed into its neighborhood by using a smoothing kernel (w)

$$w(r,h) = \begin{cases} \frac{313}{64\pi h^9}(h^2 - r^2)^3 & if\ (r < h) \\ 0 & else \end{cases} \quad (11)$$

where, r is the distance between the current particle and its neighbor, and h is the kernel support radius. The density is approximated by summing the product of the current point's mass with the kernel.

We analyzed the effect of varying the smoothing kernel [MCG03]. These kernels include the normal smoothing kernel (as given in Eq. (11)) the spiky kernel given as

$$w(r,h) = \begin{cases} \frac{15}{\pi h^6}(h - \|r\|)^3 & 0 \le \|r\| \le h \\ 0 & \|r\| > h \end{cases} \quad (12)$$

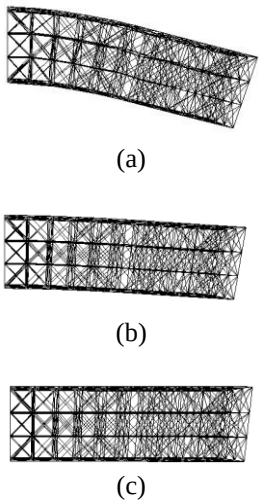
and the blobby kernel given as

$$w(r,h) = \begin{cases} \frac{15}{2\pi h^3}(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1) & 0 \le \|r\| \le h \\ 0 & \|r\| > h \end{cases} \quad (13)$$

The deformation results on a horizontal beam containing 176 points are shown in Fig. 3. Note that for all the beams shown in Fig. 3, the Young's modulus of 500,000 psi and the Poisson ratio of 0.33 are used. As can be seen, changing the smoothing kernel alters the stiffness of the soft body. This is because each kernel has a distinct support radius which influences the neighboring points. Moreover, each of these kernels has a different falloff (or, different derivative) which gives a different deformation result even though the rest of the simulation parameters are the same.

Propagation of Deformation

For propagating the stress, strain and body forces in the meshless FEM, we compute the gradient of the displacement field (U) by a moving least square interpolation between the displacement values at the current point (u_i) and its neighbor (u_j) as given by



(a)

(b)

(c)

Figure 3. Effects of different smoothing kernels on the deformation: (a) the normal smoothing kernel (Eq. 11), (b) the spiky kernel (Eq. 12) , and (c) the blobby kernel (Eq. 13)

$$e = \sum_i (u_j - u_i)^2 w_{ij} \quad (14)$$

where, w_{ij} is the kernel function given in Eq. (11).

The displacement values (u_j) are given using the spatial derivatives approximated at point (i) as

$$u_j = u_i + \nabla u.(x_j - x_i)$$

We want to minimize the error (e) in Eq. (14) so we differentiate e with respect to X, Y and Z and set the derivatives equal to zero. This gives us three equations for three unknowns

$$\nabla u\,|_x = A^{-1}\left(\sum_i (u_j - u_i)(x_j - x_i)w_{ij}\right)$$

where, $A=\Sigma_i(x_j\text{-}x_i)(x_j\text{-}x_i)^T w_{ij}$ is the moment matrix that can be pre-calculated since it is independent of the current position and displacement. Once ∇u is obtained, the strain (ε) is obtained using Eq. (2). Using this strain, the stress (σ) may be obtained using Eq. (3). The internal forces (f_{int}) in Eq. (10) are calculated as the divergence of the strain energy which is a function of the particle's volume

$$U_i = v_i \frac{1}{2}(\varepsilon_i.\sigma_i)$$

where, v_i is the volume of the particle. The force acting on neighboring particle (j) due to particle (i) is given as

$$f_j = -\nabla U_i = -v_i \frac{1}{2}\nabla\varepsilon_i.\sigma_i$$

To sum up, the internal forces acting on the particles i and j may be given as

$$\begin{aligned} f_i &= -2v_i J\sigma_i d_i \\ f_j &= -2v_j J\sigma_j d_j \end{aligned} \quad (15)$$

where, $d_i=M^{-1}(\Sigma_i(x_j-x_i)w_{ij})$ and $d_j=M^{-1}(x_j-x_i)w_{ij}$, J is the Jacobian, v is the volume of the point and σ is the stress at the given point.

Note that in the case of point masses, the volume may be calculated from the mass density in the point's neighborhood. The mass (m_i) of the point (i) is calculated using

$$m_i = sr_i^3\rho$$

where, r_i is the average distance between the mass point and its neighbors, ρ is the material density and s is a scaling constant which is calculated as

$$\beta_j = \frac{1}{\sum_j r_j^3 w_j}$$

$$s = \frac{\sum_i \beta_i}{n}$$

where, n is the total number of points. Once the mass is obtained, the per-point density is then obtained by summing the product of the mass of its neighbor (m_j) with the kernel evaluated at the neighbor j (w_j), as follow:

$$\rho_i = \sum_j m_j w_j$$

This density can then be used to obtain the volume of the point which is given as

$$vol_i = \frac{m_i}{\rho_i}$$

During the force evaluation, the internal forces are scattered in the neighborhood of the point using Eq. (15). Since scattering cannot be implemented in a GPU shader program, therefore, for parallel processing, we convert the scatter operation into a gather operation by reformulation [Buc05]. First, the sum of force matrices (F_e and F_v) is obtained

$$sumF_i = (F_e + F_v)$$

Instead of calculating the force on the point i as given in Eq. (15), we multiply the matrix ($sumF_i$) with (d_i)

$$F_i = F_i + sumF_i d_i$$

where d_i is obtained as in Eq. (15). The internal force due to the neighbor points is then given as

$$F_i = F_i - sumF_j d_j$$

where, F_i is the net internal force, j loops for each neighbor of the current point i and d_j is obtained as in Eq. (15). This allows us to run the program in parallel on all points simultaneously.

4. COUPLING BETWEEN VOLUMETRIC DEFORMATION AND RENDERING

Prior rendering algorithms have resorted to the GPGPU-based techniques for evaluating the position and/or velocity integration on the fragment shader [GEW05], [GW06], [GW08], [TE05], [VR08]. This involves rendering a screen sized quad with the appropriate textures setup and then the fragment shader is invoked to solve the integration for each fragment. The output from the fragment shader is written to another texture. On the contrary, we adopt a different approach (see Fig. 4).

We implement the meshless deformation by using the transform feedback mechanism of the modern GPU. The original unstructured mesh vertices are used directly in our implementation. Our deformable pipeline is implemented in the vertex shader stage and it outputs to the buffer object registered to the transform feedback. We use a pair of buffer objects for both the positions and velocities to avoid the simultaneous read/write race condition [ML12a]. So when we are reading from a pair of position and velocity buffers, we write to another pair. In each iteration, the pair is swapped.

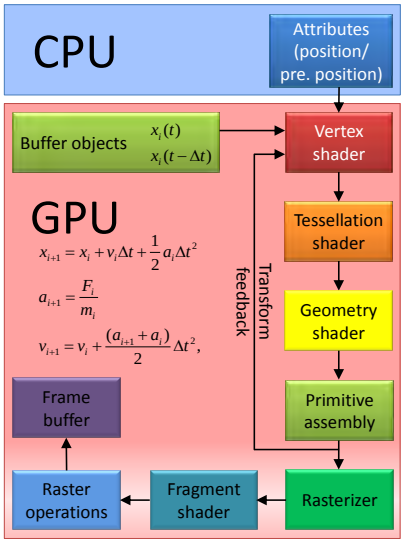


Figure 4. Proposed deformation pipeline using transform feedback

Attribute Setup for Transform Feedback

All the per-point attributes such as the current position (x_i), previous position (x^0_i), and velocity (v_i) are passed to the transform feedback vertex shader as per-vertex attributes.

The inverse mass matrices of individual nodes are stored in a texture ($texM_{inv}$) and the rest distances are stored in an attribute (r_{dist}); The point neighbor distance (r_i), support radius (h_i), the point mass (m_i) and volume (vol_i) are pre-computed and stored into a set of textures: the *neighborCountsTexture* and the *neighborListTexture*.

A pair of position and velocity buffer objects is bound as a transfer feedback buffer. This enables the vertex shader to output the results to a buffer object directly without CPU readback.

Dataflow

Referring to Fig. 5, for each rendering cycle, we swap between the two buffers to alternate the read/write pathways. Before the transform feedback can proceed, we need to bind the update array objects. Once the update array object is bound, we bind the appropriate buffer objects (for reading the current positions and velocities) to the transform feedback.

The draw point call is issued to allow the writing of vertices to the buffer object. The transform feedback is then disabled. Following the transform feedback, the rasterizer is enabled and then the points are drawn. This time, the render array objects are bound. This renders the deformed points on screen.

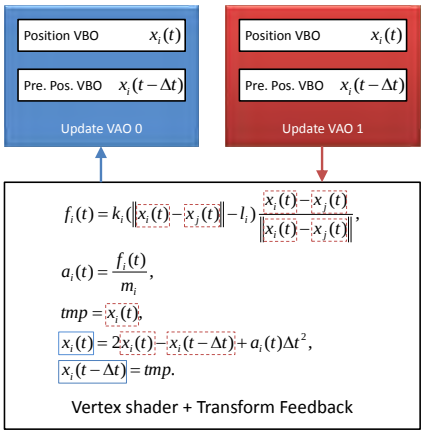


Figure 5. The vertex array object and vertex buffer object setup for transform feedback: the blue/solid rectangles show the attributes written to and the red/dotted rectangles show the attributes being read simultaneously from another vertex array object

Evaluation of Forces

The forces are evaluated per-vertex. Rather than storing the isotropic elasticity matrix (D) as a 6×6 matrix, we store it into a single vec3 attribute containing the three non-zero entries. The inverse mass matrix is pre-calculated at initialization on the CPU and then transferred to the GPU as a per-vertex attribute.

At initialization, the nearest K neighbors of the point are found using a neighborhood search. For the examples shown in this paper, the K is set as 10. This value was arrived at after some experiments. Large size of K generally makes the object stiffer. For fast and efficient search, we use a Kd-tree extracted from the given point set. The found points become the neighbors of the current point. Then, the mass, volume, weights and moment matrices are calculated for each point.

We first calculate the external forces such as the gravity force and the velocity damping force. We then calculate the Jacobians, the stresses and the internal forces using the neighbor node attributes.

Numerical Integration

Following the calculation of the forces, we perform the leap frog integration. The leap frog integration works by evaluating the velocities at 1/2 time step offset from the position. Mathematically, the leap frog integration is given as

$$x_{i+1} = x_i + v_i \Delta t + a_i \frac{\Delta t^2}{2}$$
$$v_{i+1} = v_i + \frac{a_i + a_{i+1}}{2} \Delta t \qquad (16)$$

The advantage that we obtain from this integration scheme is that it conserves the overall energy of the system. The expressions given in Eq. (16) may be converted directly into shader statements. The current position (x_i) and velocity (v_i) are passed in as per-vertex attributes whereas the next position (x_{i+1}) and velocity (v_{i+1}) are written using the transform feedback mechanism.

Cell Projection of Transformed Volume

In view of the various aspects of modeling and rendering requirements for fast rendering of transformed points, we use the hardware assisted projected tetrahedra (HAPT) algorithm [MMF10]. The deformation pipeline outputs a pair of buffer objects. These are used directly as positions for cell projection. Thus, we do not need to transfer the deformation results to CPU.

The nonrigid transformation is incorporated in the HAPT pipeline by streaming our deformed points directly. The HAPT algorithm stores positions in texture objects. In our implementation, we reuse the buffer objects output from our deformation pipeline directly. This involves no CPU readback, and thus, the data can be visualized directly.

User Interaction and Collision Detection with Deformed Volume

In a simulation system, it is often necessary to interact with the volume in realtime. In our proposed pipeline, we can interact with the volume in two ways: by modifying the vertex positions directly and by modifying the tetrahedra. In either case, we map the GPU memory to obtain the pointer to the GPU memory, index to the appropriate value and modify the value directly [ML12b].

Collision detection and response can be handled directly in the integration vertex shader by applying constraints to the calculated positions. For example,

if we have a mass point x_i, a sphere with a radius r and center C, the collision constraint may be given as

$$x_{i+1} = \begin{cases} C + \dfrac{(x_i - C).r}{|x_i - C|} & if\ |x_i - C| < r \\ x_i & else \end{cases}$$

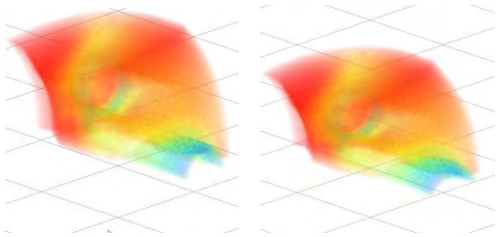
Likewise, other constraints may be integrated directly in the proposed pipeline using the vertex or geometry shader.

5. EXPERIMENTAL RESULTS AND PERFORMANCE ASSESSMENT

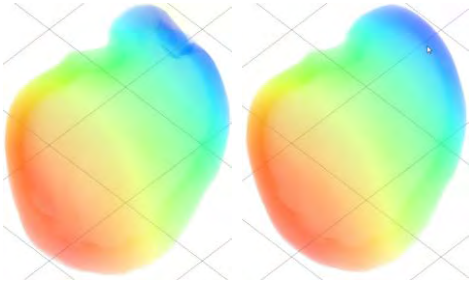
The coupled deformation and rendering pipeline has been implemented on a Dell Precision T7500 desktop with an Intel Xeon E5507 @ 2.27 MHz CPU. The machine is equipped with an NVIDIA Quadro FX 5800 graphics card. The viewport size for the renderings is 1024×1024 pixels.

The output results with deformation and rendering are shown in Fig. 6. We applied the meshless FEM to two volumetric datasets, the spx dataset (containing 2896 points and 12936 tetrahedra) and the liver dataset (1204 points and 3912 tetrahedra).

For all our experiments, the normal smoothing kernel Eq. (11) is used. We allowed the spx dataset to fall under gravity while the liver dataset was manipulated by the user. The time step value (*dt*) used for this experiment is 1/60. Thanks to the convenience of our proposed deformation pipeline, we can integrate our deformation pipeline directly into the HAPT algorithm.



(a)



(b)

Figure 6. Two frames of deformation of (a) the spx dataset falling due to gravity on the floor and (b) the liver dataset manipulated by the user

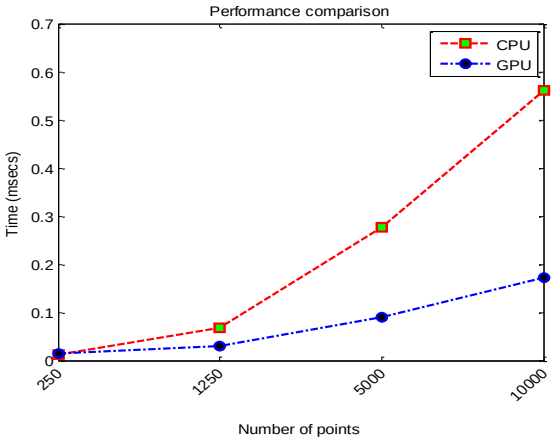


Figure 7. Performance of meshless FEM on GPU and CPU

In the second experiment, to assess the performance of the proposed method, we compare our GPU-based meshless FEM with an already optimized CPU implementation that utilized all available cores of our CPU platform. For this experiment, the bar model containing varying number of points (from 250 to 10000) was used with the time step value (*dt*) of 1/60. These timings only include the time for deformation using a single iteration and they do not include the time for rendering. These results are given in Fig. 7.
The results clearly show that the performance of the meshless FEM scales up well with the large datasets and we gain an acceleration of up to 3.3 times compared to an optimized CPU implementation. From this graph, it is clear that the runtime for CPU would be exponentially increased for larger datasets and the performance gap between the CPU and the GPU would be widened further.
It is the first time a meshless FEM model is applied to unstructured volumetric datasets. Our method uses the leap-frog integration which is semi-implicit whereas the existing mesh based FEM approaches use implicit integration schemes. The amount of time required for convergence in the case of implicit integration schemes is much more as compared to semi-implicit integration.

Moreover, the simulation system built using such formulation requires the stiffness matrix assembly which is then solved using an iterative solver such as Newton Raphson method or Conjugate Gradients (CG) method. Such stiffness assemblies are not required in meshless FEM. Therefore, ours converges much faster.

Nevertheless, for completeness, in the third experiment, we compared the performance of meshless FEM with an implicit tetrahedral FEM [ACF11]. These results are presented in Table 1.

Dataset	Tetrahedra	Frame rate (frames per second)	
		Implicit FEM	Meshless FEM
liver	3912	118.50-78.70	249.50-331.01
spx	12936	76.70-81.90	124.80-128.23
raptor	19409	40.30-43.80	71.22-71.71

Table 1. Comparison of meshless FEM against implicit tetrahedral FEM solver [ACF11]

As expected, the performance of meshless FEM is better as compared with the implicit tetrahedral FEM. Our meshless FEM is based on a semi-implicit integration scheme which does not require an iterative solver as is required for the implicit tetrahedral FEM.

6. DISCUSSION AND CONCLUSION

We have applied meshless FEM to nonrigid volumetric deformation. Using the proposed approach, interactive visualization of deformation on large volumetric dataset is made possible. We are confident of the results obtained from our experiments and would like to expand the model to address specific applications such as biomedical modeling [MCZLQS09], [MLQS09]; simulation [LSL96], [LSL97], [LSWM07]; fast ubiquitous visualization [YLS00], [ML12c]; and confocal imaging [TOTMLQS11].

We reiterate our main contributions. Firstly, we have applied meshless FEM for volumetric deformation. Secondly, we have integrated the meshless FEM into a novel deformation pipeline exploiting the transform feedback mechanism. And finally, we have integrated our novel deformation pipeline into the HAPT algorithm. There are some considerations on the meshless FEM. The deformation result is directly dependent on the number of mesh points used. More points generally give better approximation and vice versa. For better performance, we can think of two strategies. Firstly, we can use the image load-store extension in OpenGL 4.2 which allows shader programs to have access to arbitrary GPU memory. Since the hardware used in this study did not support OpenGL 4.2, this approach could not be verified. Secondly, we may use a CUDA kernel to do scattered writes, alongside a GLSL shader. This will possibly be a future research direction.

7. ACKNOWLEDGMENTS

This work is partially supported by a research grant (M408020000) from Nanyang Technological University and another (M4080634.B40) from Institute for Media Innovation, NTU. We would also like to thank the anonymous reviewers for their helpful suggestions and feedback.

8. REFERENCES

[ACF11] Allard J., Courtecuisse H., Faure F.: Implicit fem and fluid coupling on GPU for interactive multiphysics simulation. ACM SIGGRAPH 2011.

[ADLETG10] Andres D. L. C., Eliuk S., Trefftz G. H.: Simulating soft tissues using a GPU approach of the mass-spring model. Virtual Reality Conference (VR'2010), pp. 261–262, 2010.

[BBO03] Babuška I., Banerjee I., Osborn J. E., Survey of meshless and generalized finite element methods: A unified approach, Acta Numerica, 12, pp:1-125, 2003.

[Buc05] Buck I., Taking the plunge into GPU computing. Chapter 32 in GPU Gems 2, Matt Pharr and Randima Fernando (editors), 2005.

[CTA08] Comas O., Taylor Z., Allard J., Ourselin S., Cotin S., Passenger J., Efficient nonlinear fem for soft tissue modelling and its GPU implementation within the open source framework SOFA. International Symposium on Computational Models for Biomedical Simulation'08, pp. 28–39, 2008.

[DGW10] Dick C., Georgii J., Westermann R., A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. In Simulation Modelling Practice and Theory'10, 19, No. 2, pp. 801–816, 2010.

[GEW05] Georgii J., Echtler F., Westermann R.: Interactive simulation of deformable bodies on GPUs. In Simulation and Visualization'05, 2005.

[GW05] Georgii J., Westermann R.: A multi-grid framework for real-time simulation of deformable volumes. Workshop on Virtual Reality Interactions and Physical Simulations'05, 2005.

[GW06] Georgii J., Westermann R., A generic and scalable pipeline for GPU tetrahedral grid rendering. IEEE Transaction on Visualization and Computer Graphics'06, 2006.

[HF00] Huerta A., Fernandez M. S., Enrichment and coupling of the finite element and meshless methods, International Journal for Numerical Methods in Engineering, 48, No. 11, pp:1615–1636, August 2000.

[LSL97] Lin F., Seah H. S., Lee Y. T., Structure Modeling and Context-Free-Grammar: Exploring a new approach for surface construction. Computers and Graphics (1997), 21, No. 6, pp. 777–785, 1997.

[LSL96] Lin F., Seah H. S., Lee Y. T., Deformable volumetric model and isosurface: Exploring a new approach for surface construction. Computers and Graphics (1996), 20, No. 1, pp. 33–40, 1996.

[LSWM07] Lin F., Seah H. S., Wu Z., Ma D., Voxelisation and fabrication of freeform models. Virtual and Physical Prototyping'07, 2 No. 2, pp. 65–73, 2007.

[MCG03] Mueller M., Charypar D., Gross M.: Particle based fluid simulation for interactive applications. In Proceedings of the ACM SIGGRAPH Symposium on Computer Animation (SCA'03), pp. 154–159, 2003.

[MCZLQS09] Movania M. M., Cheong L. S., Zhao F., Lin F., Qian K., Seah H. S., "GPU-based Surface Oriented Interslice Directional Interpolation for Volume Visualization," The 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL'09), Bratislava, Slovak Republic, November 24-27, 2009.

[MG04] Mueller M., Gross M., Interactive virtual materials. Proceedings of Graphics Interface (GI'04), pp. 239–246, 2004.

[MHSS04] Mosegaard J., Herborg P., Sangild Sorensen T.: A GPU accelerated spring mass system for surgical simulation. Health Technology and Informatics'04, pp. 342–348, 2004.

[MJLW07] Miller K., Joldes G., Lance D., Wittek A., Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. Communications in Numerical Methods in Engineering'07, 23, No. 1, pp. 801–816, 2007.

[ML03] Mendoza C., Laugier C., Simulating soft tissue cutting using finite element models. IEEE International Conference on Robotics and Automation'03, pp. 1109–1114, 2003.

[ML12a] Movania M. M., Lin F., A novel GPU-based deformation pipeline. ISRN Computer Graphics, vol. 2012 (2012), p. 8.

[ML12b] Movania M. M., Lin F., Real-time physically-based deformation using transform feedback. Chapter 17 in The OpenGL Insights, Christophe, Riccio and Patrick, Cozzi (Ed.), AK Peters/CRC Press, 2012, pp. 233-248.

[ML12c] Movania M. M., Lin F., High-Performance Volume Rendering on the Ubiquitous WebGL Platform, 14th IEEE International Conference on High Performance Computing and Communications (HPCC'12), Liverpool, UK, 25-27 June, 2012.

[MLQS09] Movania M. M., Lin F., Qian K., Seah H. S., "Automated Local Adaptive Thresholding for Real-time Feature Detection and Rendering of 3D Endomicroscopic Images on GPU," The 2009 International Conference on Computer Graphics and Virtual Reality (CGVR'09), Las Vegas, US, July 13-16, 2009.

[MMF10] Maximo A., Marroquim R., Farias R., Hardware assisted projected tetrahedra. Computer Graphics Forum, 29, No. 3, pp: 903–912, 2010.

[NMK06] Nealen A., Mueller M., Keiser R., Boxermann E., Carlson M., Physically based deformable models in computer graphics. In STAR Report Eurographics 2006 vol. 25, pp. 809–836, 2006.

[NRBD08] Nguyena V. P., Rabczukb T., Bordasc S., Duflotd M., Meshless methods: A review and computer implementation aspects, Mathematics and Computers in Simulation, 79, No. 3, pp:763–813, December 2008.

[SB09] Sampath R., Biros G., A parallel geometric multigrid method for finite elements on octree meshes. In review, available online accessed in 2012 http://www.cc.gatech.edu/grads/r/rahulss/, 2009.

[ST99] Shapiro V., Tsukanov I., Meshfree Simulation of Deforming Domains," Computer Aided Design, 31, No. 7, pp: 459–471, 1999.

[TCO08] Taylor Z., Cheng M., Ourselin S., High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. IEEE Trans. Medical Imaging, vol. 27, pp. 650–663, 2008.

[TE05] Tejada E., Ertl T., Large steps in GPU-based deformable bodies simulations. In Simulation Modeling Practice and Theory, 13 No. 8, Elsevier, pp. 703–715, 2005.

[TOTMLQS11] Thong P. S. P, Olivo M., Tandjung S. S., Movania M. M., Lin F., Qian K., Seah H. S., Soo K. C., "Review of Confocal Fluorescence Endomicroscopy for Cancer Detection," IEEE Photonics Society (IPS) Journal of Selected Topics in Quantum Electronics, Vol. PP, Issue 99, 2011. DOI: 10.1109/JSTQE.2011.2177447.

[VR08] Vassilev T., Rousev R., Algorithm and data structures for implementing a mass-spring deformable model on GPU. Research and Laboratory University Ruse 2008 (2008), pp. 102–109, 2008.

[YLS00] Yang Y. T., Lin F. and Seah H. S., "Fast Volume Rendering," Chapter 10 in Volume Graphics, Springer, January 2000.

[ZWP05] Zhong H., Wachowiak M., Peters T.: A real time finite element based tissue simulation method incorporating nonlinear elastic behavior. Computer Methods Biomechan. Biomed. Eng., 6, No. 5, pp. 177–189, 2005.

9. APPENDIX

$$\int_{V^e}(B\delta u^e)^T\sigma dV=(\delta u^e)^T q^e+\int_{A^e}(N\delta u^e)^T tdA$$

$$(\delta u^e)^T\int_{V^e}B^T\sigma dV=(\delta u^e)^T(q^e+\int_{A^e}N^T tdA)$$

$$\int_{V^e}B^T\sigma dV=q^e+\int_{A^e}N^T tdA$$

Substitution using Eq. (3)

$$\int_{V^e}B^T D\varepsilon dV=q^e+\int_{A^e}N^T tdA$$

$$\int_{V^e}B^T DBu^e dV=q^e+\int_{A^e}N^T tdA$$

$$B^T DBu^e\int_{V^e}dV=q^e+\int_{A^e}N^T tdA$$

$$B^T DBu^e V^e=q^e+\int_{A^e}N^T tdA$$

On Global MDL-based Multichannel Image Restoration and Partitioning

Tetyana Ivanovska
University of Greifswald, Germany
tetyana.ivanovska@uni-greifswald.de

Horst K. Hahn
Fraunhofer MEVIS, Germany
horst.hahn@mevis.fraunhofer.de

Lars Linsen
Jacobs University, Germany
l.linsen@jacobs-university.de

ABSTRACT

In this paper, we address the problem of multichannel image partitioning and restoration, which includes simultaneous denoising and segmentation processes. We consider a global approach for multichannel image partitioning using minimum description length (MDL). The studied model includes a piecewise constant image representation with uncorrelated Gaussian noise. We review existing single- and multichannel approaches and make an extension of the MDL-based grayscale image partitioning method for the multichannel case. We discuss the algorithm's behavior with several minimization procedures and compare the presented method to state-of-the-art approaches such as Graph cuts, greedy region merging, anisotropic diffusion, and active contours in terms of convergence, speed, and accuracy, parallelizability and applicability of the proposed method.

Keywords: Segmentation, Denoising, Minimum Description Length, Energy Minimization, Multichannel images.

1 INTRODUCTION

The goal of image partitioning is to detect and extract all regions of an image which can be distinguished with respect to certain image characteristics. A special form of image partitioning is image segmentation, where one or a few regions of given characteristics are separated from the rest of the image. If the underlying image is supposed to be piecewise constant, image partitioning is equivalent to the restoration of that image, which is often obtained using denoising algorithms. Although there exist plenty of methods for solving image partitioning, segmentation, and restoration problems, many of them are task-specific or require intensive user interaction and triggering of a large number of parameters.

In this paper, we restrict ourselves to generic solutions using automatic methods based on energy minimization. Hence, two tasks need to be tackled: First, one needs to formulate an energy functional based on some assumptions and, second, one needs to find an appropriate and computationally feasible minimization algorithm. The main emphasis of this work is put on analysis and discussion of an energy functional, based on the assumption that the processed image is multichannel, piecewise constant, and affected by white Gaussian noise. We analyse and compare several minimization procedures and draw analogies to other approaches. Color images are used as examples, as they document the algorithmic behavior in an intuitive manner. However, all steps work are applicable for any multichannel image data. The paper is organized as follows. In Section 2 the related work in this area is described. In Section 3 we formulate the energy functional and describe two minimization procedures. Our findings are presented and discussed in Section 4.

2 RELATED WORK

Global energy minimization approaches originate from such seminal works as the ones by Mumford and Shah [24] and Blake and Zisserman [2]. The Markov Random Fields (MRF) framework (see the seminal work of Geman and Geman [8]) is a stochastic branch of the energy minimization approaches. In the last years a great breakthrough has been done in the direction of Markov random fields methods [19]. Such methods as Graph Cuts [5] are mathematically well described and allow to find a solution that lies close to the global optimum.

The minimum description length (MDL) based approach [15] uses basic considerations from information theory [19] in the formulation of the energy term, in the sense that the best image partitioning is equivalent to obtaining the minimum description length of the image with respect to some specific description language.

There exist two main tendencies in further developement of this approach. The first one consists in extending the functional to be minimized. These extentions include either different noise models (not only white Gaussian noise), or elimination of the model parameters that must be manually tuned by a user.

Kanungo [12] et al. formulated the functional for multiband and polynomial images. Lee [17] considered correlated noise. Galland [7] et al. considered speckle, Poisson, and Bernoulli noise. Zhu and Yuille [31] proposed an algorithm combining region growing, merging, and region competition which permits one to segment complex images. In several approaches [20, 16], an extended version of the functional is used and all user-defined parameters are eliminated. For example, Luo and Khoshgoftaar [20] propose to use the well known Mean-Shift method [6] to obtain the initial segmentation and start the MDL-based region merging procedure from it.

These approaches utilize region growing, as the minimization of the extended functional is infeasible. However, such an iterative technique, generally, does not lead to a stable local minimum and can give much coarser results, when compared to the relaxation technique, if the procedure of region merging is non-reversible. Moreover, the authors state, for instance, in [20], that the initial region selection has a strong impact on the efficiency and effectiveness of the region merging.

The second direction is to use a simplified or limited model with some user-defined parameters, but apply a global minimization procedure, which allows one to find at least a stable local minimum. Kerfoot and Bresler [13] formulated a full MDL-based criterion for piecewise constant image partitioning, but the class of images is limited to the class of simply-connected ones. For minimization such methods as Graph Cuts [5] have gained in popularity in the last years.

3 METHODS

3.1 Multichannel Model Description

The fundamental idea behind the Minimum Description Length (MDL) [19, 23] principle is that any regularity in the given data can be used to compress the data.

The image partitioning problem with respect to the MDL principle can be formulated as follows: Using a specified descriptive language, construct the description of an image (code) that is simplest in the sense of being shortest (when coded, needs the least number of bits) [15]. Let $L(M)$ denote the language for describing a model M and $L(D|M)$ the language for describing data D given model M. Moreover, let $|.|$ denote the number of bits in the description. The goal is to find the model M that minimizes the code length $C_l = |L(M)| + |L(D|M)|$. This corresponds to the two-part MDL code [9]. If the a priori probabilities $P(M)$ of the described models are known, then the number of bits in the description equals the negative base-two logarithm of the probability of the described models [19]: $|L(M)| = -\log_2 P(M)$.

In terms of image partitioning and restoration the code length can be written as $C_l = |L(u)| + |L(z-u)|$,where the model we are looking for is the underlying image representation (or partitioning) u that minimizes the code length. The term z describes the initial (or given) image, and the difference $r = (z-u)$ between the given image z and the partitioning u corresponds to the noise in the image. The noise describes the data with respect to model u.

A simple implementation of the MDL principle for image partitioning was presented by Leclerc [15, 23]: he assumed a piecewise constant model and derived the functional (or energy term)

$$C_l = \frac{b}{2}\sum_{i\in I}\sum_{j\in N_i}\left(1-\delta\left(u_i-u_j\right)\right) + a\sum_{i\in I}\left(\frac{z_i-u_i}{\sigma}\right)^2, \quad (1)$$

where u denotes the underlying image, z the given image, and σ^2 the noise variance. Moreover, $\delta(u_i-u_j)$ denotes the Kronecker delta (1 if $(u_i = u_j)$, else 0), I denotes the range of the image, and N_i is the neighbourhood of the ith pixel, a and b are constants. The first term in Equation (1) encodes the boundaries of the regions, whereas the second term encodes the noise in form of uncorrelated white Gaussian noise.

One can observe the similarities between the functional in Equation (1) with constants a, b, σ and the energy considered in different MRF approaches, namely in the Graph Cut methods (see [5] for more details): $E(f) = \lambda \sum_{(p,q)\in N} V_{p,q}(f_p, f_q) + \sum_{p\in I} D_p(f_p)$, where the interaction potential V between pixels p, q having labels (colors) f_p, f_q is taken from the Potts model (which corresponds to the Kronecker deltas), D is the distance between the initial and current colors of the pixel p (which corresponds to the noise values), and λ is a constant.

We expand on this approach to derive a multichannel image description length. For encoding the model, i.e., deriving $L(u)$, we have to encode the boundaries of the regions. To do so, we calculate the number of pixels that contribute to the boundary. Hence, the codelength for the boundary encoding is given by $|L(u)| = \frac{b}{2}\sum_{i\in I}\sum_{j\in N_i}\left(1-\prod_{k\in Ch}\delta\left(u_i^k-u_j^k\right)\right)$, where k denotes the channel, Ch is the range of channels (e.g., RGB in the three-channel color case), and other notations are as above. To encode the data that do not fit the model, i.e., the noise, we derive $L(z-u)$ assuming that the values in each channel are subject to white Gaussian noise with parameters $(0, (\sigma^k)^2)$. This assumption implies that the noise between channels is not correlated. Such an assumption allows for better understanding of the underlying processes and is often sufficient for many applications, since the uncorrelated noise can appear during the transmission, storing, or manipulation of images [1]. Moreover, the assumption holds when a multichannel image is combined from different independent modalities for processing.

The codelength of the noise is derived as

$$|L(z-u)| = -\sum_{i\in I}\sum_{k\in Ch}\log_2 P(r_i^k) \qquad (2)$$
$$= -\sum_{i\in I}\sum_{k\in Ch}\log_2\left(\frac{q}{\sqrt{2\pi(\sigma^k)^2}}\exp\left(-\frac{(r_i^k)^2}{2(\sigma^k)^2}\right)\right)$$
$$= \frac{1}{2\ln 2}\sum_{i\in I}\sum_{k\in Ch}\left(\frac{r_i^k}{\sigma^k}\right)^2 + const$$

where $q=1$ is the pixel precision, *const* is an additive constant, which is discarded, if $\left(\sigma^k\right)^2$ is considered constant and equal for all channels.

The resulting codelength functional becomes $C_l = \frac{b}{2}\sum_{i\in I}\sum_{j\in N_i}\left(1-\prod_{k\in Ch}\delta\left(u_i^k-u_j^k\right)\right) + \frac{1}{2\ln 2}\sum_{i\in I}\sum_{k\in Ch}\left(\frac{r_i^k}{\sigma^k}\right)^2$. This functional corresponds to the one considered in the Graph cuts method up to the constant weights.

3.2 Minimization

Having formulated the energy functionals, one needs to minimize them for a given image in order to compute the image partitioning. As it has been shown that computing the global optimum even of the simplest functional is an NP-hard problem [5], in practice one has to look for efficient approximations for it.

In the current paper, we will deal with two optimization approaches for energy minimization: a discrete one and a continuous one, namely, the α-expansion Graph Cut algorithm, introduced by Boykov [5], and the GNC-like approach, introduced by Leclerc [15], with several modifications, which have been done to check the convergence.

Graph cuts is an efficient minimizing technique that allows for finding a local minimum within a known factor of the global one. This algorithm belongs to the class of discrete optimization. Here, we give the outline of the algorithm and refer the reader to the original paper by Boykov et al [5] for further details. Let S and $\mathscr{L}$ denote image pixels (lattice) and the palette (set of all possible colors), correspondingly. The labeling u is described as $\{\mathscr{S}_l | l\in\mathscr{L}\}$, where $\mathscr{S}_l = \{p\in S | u_p = l\}$ is a subset of pixels with assigned color l. Given a label α a move from a labeling u to a new labeling u' is called an α- expansion if $\mathscr{S}_\alpha \subset \mathscr{S}'_\alpha$ and $\mathscr{S}'_l \subset \mathscr{S}_l$ for any label $l\neq\alpha$. In other words, an α-expansion move allows any set of image pixels to change their labels to α [5]. The minimum of the energy E for each label α is found by constructing a graph and finding the minimum cut for it. It is efficiently done by the algorithm developed by Boykov and Kolmogorov [3].

Start with an arbitrary partitioning u
repeat
  Set *success* $\leftarrow 0$
  for all $\alpha\in\mathscr{L}$ **do**
    Find $\hat{u} = \arg\min E(u')$ among u' within one α-expansion of u
    if $E(\hat{u}) < E(u)$ **then**
      $u \leftarrow \hat{u}$
      success $\leftarrow 1$
    end if
  end for
until *success* $\neq 0$
Return u

A Relaxation method using ideas of Graduated Non Convexity (GNC) by Blake and Zisserman [2] was proposed by Leclerc [15]. This is a continuous optimization method, where the labeling u is not selected from the given palette, as in the Graph Cuts case, but is iteratively computed. Here, $u\in\mathbb{R}^n$. The basic concept of the minimization procedure is to replace the non-convex codelength functional $C_l(u)$ by an embedding in a family of continuous functions $C_l(u,s)$, where $s\in\mathbb{R}$ is a user-defined parameter, that converge towards the target functional $C_l(u)$ when s goes to zero. $\lim_{s\to 0} C_l(u,s) = C_l(u)$. For the starting value of s, the functional $C_l(u,s)$ is convex such that standard convex minimization procedures can compute the single minimum. When s approaches zero, number and positions of the local minima of $C_l(u,s)$ become those of C_l. The minimization procedure iterates over s, which steadily decreases, and minimizes $C_l(u,s)$ for the respective value of s in each iteration step.

To obtain a continuous embedding, the discontinuous parts in functional C_l need to be replaced by a continuous approximation. The discontinuity of C_l is due to the use of the function δ. Hence, function δ is replaced by a continuous approximation that converges to δ when s goes to zero [15]. We use the approximation $\delta\left(u_i^k-u_j^k\right)\approx\exp\left(-\frac{\left(u_i^k-u_j^k\right)^2}{\left(s\sigma^k\right)^2}\right) = e_{ij}^k$.

The minimization iteration starts with a sufficiently large value $s=s^0$ and computes the (global) minimum u^0 of the convex functional $C_l(u,s^0)$. In each iteration step $T+1$, we set $s^{T+1} = rs^T$, where $0<r<1$, and compute the local minimum of $C_l(u,s^{T+1})$ starting from minimum u^T of the previous iteration step. The iteration is repeated until s is sufficiently small, i.e., until $s<\varepsilon$ with ε being a small positive threshold.

To compute the local minimum u on each iteration we apply Jacobi iterations [26]. The functional $C_l(u,s)$ is convex, if we choose a value for s that satisfies $(x_i^k - x_j^k)^2 \le 0.5\left(s\sigma^k\right)^2$ for all pixels i and j with $i\neq j$) and all channels k. Hence, when this condition is met, the local minimum must be a global one. The condition needs to be fulfilled for the starting value $s=s^0$. Then, the condition $\frac{\partial C_l(u,s^T)}{\partial u_i^k} = 0$for the local minimum at iteration step T becomes

$$\frac{2a\left(u_i^k-z_i^k\right)}{\left(\sigma^k\right)^2} + \frac{b}{2}\sum_{j\in N_i}\left[\frac{2\left(u_i^k-u_j^k\right)}{\left(s^T\sigma^k\right)^2}\prod_{l\in Ch} e_{ij}^l\right] = 0, \qquad (3)$$

where constant $a = (2\ln 2)^{-1}$. As Equation (3) cannot be solved explicitly, we use an iterative approach, where at each iteration step $t+1$ we compute

$$u_i^{k,t+1} = \frac{z_i^k + \frac{b}{a(s^T)^2} \sum_{j \in N_i} u_j^{k,t} \prod_{l \in Ch} e_{ij}^{lt}}{1 + \frac{b}{a(s^T)^2} \sum_{j \in N_i} \prod_{l \in Ch} e_{ij}^{lt}}. \quad (4)$$

In total, we have two nested iterations. The outer iteration denoted by T iterates over s^T, while the inner iteration denoted by t iterates over $u_i^{k,t+1}$. Considering the behavior of the exponential function, the termination criterion for the inner loop is given by $\left| u_i^{k,t+1} - u_i^{k,t} \right| < s^T \sigma^k, \quad \forall i \in I$. Starting with $u = z$, the minimization procedure can be summarized by the following pseudocode:

while $s \geq \varepsilon$ **do**
  start with local minimum for u found in previous iteration
  while termination criterion for u is not met **do**
    recalculate u using Equation (4)
  end while
  update s
end while

Comparison to Anisotropic Diffusion. The derived iterative scheme for computing u_i^{t+1} in the single-channel case, cf. [15], is similar to the iteration in the well-known anisotropic diffusion approach. The continuous form of the Perona-Malik equation [25] for anisotropic diffusion is given by

$$\frac{\partial I}{\partial t} = div(g(\|\nabla I\|) \cdot \nabla I), \quad (5)$$

where I denotes the image and function g is defined by $g(\|\nabla I\|) = \exp\left(-\left(\frac{\|\nabla I\|}{K}\right)^2\right)$ with flow constant K. The discrete version of Equation (5) is given by $I_i^{t+1} - I_i^t + \lambda \sum_{j \in N_i} \left(I_i^t - I_j^t\right) \exp\left(-\left(\frac{I_i^t - I_j^t}{K}\right)^2\right) = 0$, where λ is a normalization factor.

The local minimum of the MDL-based energy C_l in single-channel notation (for a fixed s) is defined by solving $\frac{\partial C_l(u,s)}{\partial u_i} = 0$, which leads to

$$(u_i - z_i) + \alpha \sum_{j \in N_i} \left[(u_i - u_j) \exp\left(-\frac{(u_i - u_j)^2}{(s\sigma)^2}\right)\right] = 0.$$

The continuous version of this equation can be written as

$$\int_0^{t_{end}} \frac{\partial u}{\partial t} dt = div(g(\|\nabla u\|) \cdot \nabla u), \quad (6)$$

where $u_0 = z$ and $u_{t_{end}}$ describes the image values at the current step t_{end}. Comparing the continuous form of the Perona-Malik equation (5) with the continuous form of the MDL-based equation (6), one can immediately observe the similarity. The main difference is the integral on the left-hand side of Equation (6). The integral represents the changes between the current image and the initial image. Thus, this version of the MDL-based minimization algorithm can be considered as an "anisotropic diffusion algorithm with memory".

In general, the assumptions about the convexity of $C_l(u,s)$ do not hold anymore, when s is small. Hence, the relaxation method with Jacobi iterations can give an inadequate result. To check this we implemented the steepest descent minimization scheme [26] and compared the results.

4 RESULTS AND DISCUSSION

The presented algorithm belongs to the class of algorithms that are not purely for denoising or segmentation, but can be treated as an elegant combination of both approaches. Usually, some part of meaningful image data might be lost on the denoising step, which affects the subsequent segmentation results. Here, on the contrary, the complete information is used both for boundary preservation and noise exclusion, which is referred in the literature as image restoration [28] or reconstruction [25].

The algorithm (minimization procedures with Jacobi iterations, Gradient descent, and Graph cuts) was implemented using C/C++ programming language, optimized appropriately, and was compiled under gcc4.4. The Graph cuts code for grayscale images was kindly provided on the Vision.Middlebury web-site [22].

Figure 1 documents the general behavior of the Relaxation scheme. Starting with a synthetic 100×100 image with manually added noise, we apply the Jacobi iteration minimization procedure. The three rows in Figure 1 show (intermediate) results at iteration $T = 4$, $T = 600$, and $T = 1484$, respectively. The first column shows the currently detected underlying image and the second column the currently removed noise, i.e., the difference between the initial image and the currently detected underlying image (first column of Figure 1). For the generation of the images in column three, we picked one row of the image, i.e., a horizontal cut through the image, and show the current values of u for that single row. The third column in Figure 1 documents nicely, how individual values (first row) start assimilating and grouping together (second row) to eventually form a piecewise constant representation (third row).

To qualitatively describe the restoration results, we evaluated the difference between the resulting and initial (not noisy) images using two measures. The first one is the mean squared error (MSE) estimator [18], which is suitable to show whether the colors were reconstructed as close to

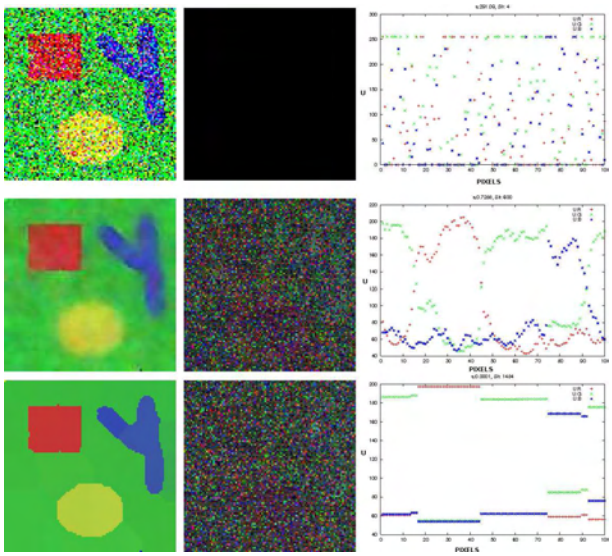


Figure 1: Different phases of MDL minimization on timesteps 4, 600, and 1484. The images from left to right are: the detected underlying image, the removed noise, image values u of one row (The pixel coordinates are given in x axis, and u values are in y axis).

the initial ones as possible, i.e., for the denoising part. For the multichannel case, MSE is defined as $MSE = \frac{1}{cwh}\sum_{c_1\in c}\sum_{y\in h}\sum_{x\in w}(In(x,y,c_1)-Res(x,y,c_1))^2$, where c is the number of color channels, w is the image width, h is the image height, and In and Res define the initial and resulting image, respectively.

The second metric is utilized to evaluate the segmentation results, i. e., oversegmentation and inaccurate boundary localization. It is based on the one proposed by Mezaris et al [21]. This metric is defined as follows. Let $S = s_1, s_2, ..., s_K$ be the segmentation mask to be evaluated, comprising K regions s_k, $k = [1,K]$, and let $R = r_1, r_2, ..., r_Q$ be the reference mask, comprising Q reference regions r_q, $q = [1,Q]$. Each region r_q is associated with a different region s_k, i. e. s_k is chosen such that the overlap $r_q \cap s_k$ is maximized. Let $A(r_q, s_k)$ denote the set of region pairs and let N_S denote the set of non-associated regions of mask S. Energy E_b is used for the evaluation, values closer to zero indicate better segmentation: $E_b = \sum_{q=1}^{Q} E_q + \sum_{s_k \in N_S} F_k$, where E_q and F_k are defined are follows.

$$E_q = \sum_{p\in(r_q - r_q\cap s_k)} f_1(p,r_q) + \sum_{p\in(s_k - s_k\cap r_q)} f_2(p,r_q) \quad (7)$$

$$F_k = \alpha \sum_{p\in s_k} f_1(p,r_q) \quad (8)$$

E_q is a weighted sum of misclassified pixels for region pair (r_q, s_k) $\forall (r_q, s_k) \in A$. F_k is a weighted sum of misclassified pixels $\forall s_k \in N_S$. f_1 and f_2 are weight functions, proposed by Villegas et al [30] to deal with the fact that the distance of a misclassified pixel from the boundary of the reference region to which it belongs affects the visual relevance of the error [21]. f_1 is used for false negatives and f_2 is used for false positives.

$$f_1(p,r_q) = d(p,r_q) * 10^{-4},$$
$$f_2 = \begin{cases} d(p,r_q) * 10^{-4} & d(p,r_q) < 10 \\ 10^{-3} & otherwise \end{cases}.$$

Moreover, d is the Euclidean distance between the pixel p and the boundary of the region r_q. α is a weight parameter which was heuristically set to 100 in our experiments, since we would like to penalize oversegmentations.

First, we run a series of tests on synthetic images with artificially added Gaussian noise applying three algorithms: Graph Cuts, Relaxation with Jacobi iterations, and Relaxation with steepest descent. Figures 2 and 3 demonstrate the high-quality results produced by both Graph Cuts and Relaxation with gradient descent methods. The exact regions are reconstructed with few misclassifications. Graph Cuts restore the closest colors to the initial image ($MSE = 830.65$).

However, the computational costs of these methods are rather high. For the Graph cuts method the execution time is dependent on the palette size, i. e., the number of colors (labels) which would be considered for expansion. Ideally, the palette should include the whole color space. In this case, the optimum will be found accurately. Boykov et al. proposed to reduce the space of labels for this case [4], taking, for instance, only unique colors of the image and their closest neighbors as labels. Such a choice allows for a massive label space reduction, but leaves enough variability for the label expansion. In our experiments the computation for a 100×100 color image with $50,000$ labels (unique colors and their closest neighbours of a noisy synthetic image) took around 10 minutes per iteration. As usually several iterations are needed, this approach appears to be infeasible for bigger datasets.

The relaxation scheme with the Gradient Descent method for a 100×100 color image takes in several hours, which makes this method inapplicable for real datasets.

The relaxation scheme with Jacobi iterations restores more regions than required, however, they have very close colors and the "weak" region boundaries can not be distinguished by a human eye. Due to such color closeness, these regions can be merged into one using a simple region-merging procedure. Starting at any pixel (seed point), the algorithm subsequently adds neighbouring pixels to the region if the distance in color space between them and the region color is lower than a certain threshold. The region color is the average color of the pixels that belong to it. The algorithm stops when all pixels belong to some regions. Computational costs of the region merging procedure are negligible

when compared to the ones of the Relaxation method. When the simple region merging is applied, we obtain $E_b = 0.0021$, which is slightly better than the other results.

The main advantage of the Jacobi iteration approach is its independence of the palette size and straightforward parallelizability, such that computations only take seconds [11], which makes this method attractive for real applications.

Next, we would like to compare the presented method to other (similar) approaches, namely, anisotropic diffusion, region merging, and a combination of these two procedures. Although anisotropic diffusion is primarily used to remove noise from digital images without blurring edges, it can be used in edge detection algorithms. By running the diffusion with an edge seeking diffusion coefficient for a certain number of iterations, the image can be evolved towards a piecewise constant image with the boundaries between the constant components being detected as edges [25]. We compared our approach to the anisotropic diffusion algorithm for multichannel images, presented by Sapiro and Ringach [27]. This method similarly to ours considers the influence of all image channels at once, which allows for better boundary preservation when compared to the methods treating each image channel separately. However, our approach has a "memory", as it always refers to the initial image instead of iterating from the result obtained on the previous time step. Such a behavior should guarantee a better boundary preservation and initial color restoration if the parameters are properly chosen. Figure 4 demonstrates that Sapiro's method successfully eliminates the noise (except from some mistakes on the image boundaries) and preserves most of the object boundaries. Our approach appears to be stable to noise, preserves object boundaries, restores the colors close to the initial ones, and produces the most uniformly colored background.

Due to the energy functional complexity, which arises for the models with no manually adjustable parameters or for more complicated noise models, it is often difficult to find a feasible minimization procedure. Thus, many authors follow a "greedy" region merging strategy [12, 20, 14, 16]. The idea of the algorithm is as follows. It starts from some initial oversegmentation. Then at each timestep it chooses two neighbouring regions and merges them to form a new region. These two regions are chosen in such a way that, when they are merged, it provides the largest energy reduction amongst all other possible merges. Although this procedure is very fast, it has several drawbacks. First, it does not guarantee convergence to a stable local minimum, especially, in the presence of noise. Second, the way the initial oversegmentation is chosen also affects the results.

Although Kanungo et al. [12], Luo and Khoshgoftaar [20], Lee [16], and Koepfler et al. [14] propose to use different functionals for color image segmentation, the region merging procedures are similar to each other. We assume that a more complicated functional could lead to a more adequate result. However, there is no guarantee about the convergence to a stable local minimum, and this issue stays unresolved.

We compared our approach to the algorithm proposed by Koepfler et al. [14]. They use the Mumford-Shah functional for the piecewise constant case, which coincides with the MDL-based functional C_l. Here, we computed both metrics to evaluate the results. As it can be observed in Figure 5, the iterative region merging produces results with some misclassified boundaries, which appears due the fact that the merging algorithm did not land close to the minimum, and not completely erased noise when compared to Figure 4. The relaxation approach can be approximated with a pipeline that utilizes several algorithms, namely, denoising, boundaries sharpening, and region merging. We constructed a pipeline from Sapiro's anisotropic diffusion and Koepfler's region merging. As it can be observed in Figure 5, the pipeline produces the best result in terms of boundary preservation and noise elimination when compared to the independent application of these methods. However, in general, each method in the pipeline requires additional adjustment of the parameters for each image.

Our comparison shows that the proposed approach produces the best results in terms of color (the lowest $MSE = 377.628$) and region restoration (the lowest $E_b = 0.0008$) for the test image.

We also compared our results to the results obtained with the Active contours without edges [29] method introduced by Chan and Vese. This is a variational approach based on energy minimization, and the energy is formulated using the Mumford-Shah functional for the piecewise constant case. For our tests, we applied the 4-phase version of the algorithm for color piecewise constant images with the parameters for each phase:$\lambda_{1,2} = 1$, $\nu = 0$, as it is recommended by the authors. We experimented with different values of parameter μ and initial contour locations. We executed the algorithm with max. 2000 iterations with the $timestep = 0.5$. Figure 6 documents that it is problematic to obtain decent results for images with low signal-to-noise ratio for the reasonable amount of time and the prior denoising is needed.

When applied to real images, our approach allows for obtaining the results with different levels of detail, which reminds one of the multi-scale theory of piecewise image modeling, introduced by Guigues et al. [10]. In Figure 7 the results illustrate this effect. Increasing the value of b allows us to reduce the

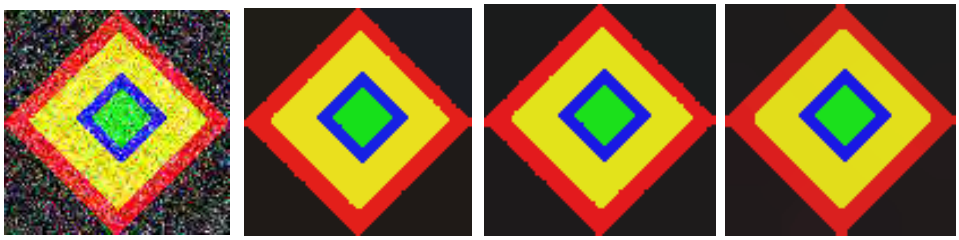


Figure 2: *First*: Input diamand image with manually added noise $N(0,70^2)$. *Second*: Graph cuts result. $MSE = 830.65$. $E_b = 0.0027$ *Third*: Relaxation (Steepest descent on inner iterations) result. $MSE = 2273.8096$. $E_b = 0.0031$ *Fourth*: Relaxation (Jacobi iterations) result. $MSE = 2418.262$. No region merging $E_b = 651.287$. With region merging $E_b = 0.0021$

Figure 3: *First*: True image with 8 regions (5 unique colors). *Second*: Piecewise-constant regions are marked with random different colors on the Graph cuts result. The image consists of 8 regions (7 unique colors). *Third*: Regions are marked with random different colors on the Relaxation (Descent) result. The image consists of 8 regions. *Fourth and Fifth*: Piecewise-constant regions are marked with random different colors on the Relaxation (Jacobi) result. The image consists of 49 regions (before region merging) and 8 regions (after region merging).

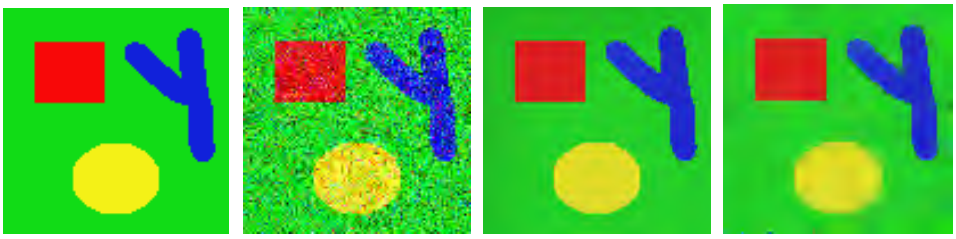


Figure 4: *First*: Initial synthetic image. *Second*: Synthetic image with added noise $N(0,70^2)$. *Third*: Result for our approach with subsequent region merging. $MSE = 377.628$. $E_b = 0.0008$. *Fourth*: Result for Sapiro's anisotropic diffusion with parameters: $edge = 49$, $numStep = 45$. $MSE = 447.83$.

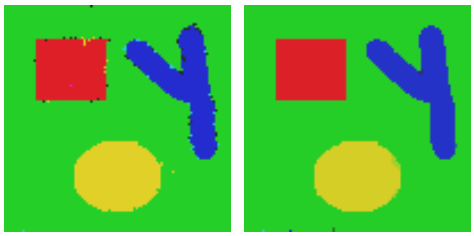


Figure 5: *First*: Result for iterative region merging with $\lambda = 10^6$. Noise is not erased completely. Further increasing the parameter v_0 does not improve the result. $MSE = 436.538$. $E_b = 6.5369$ *Second*: Result for the pipeline consisting of the anisotropic diffusion and region merging based on the Mumford-Shah functional. $MSE = 419.7864$. $E_b = 0.26$

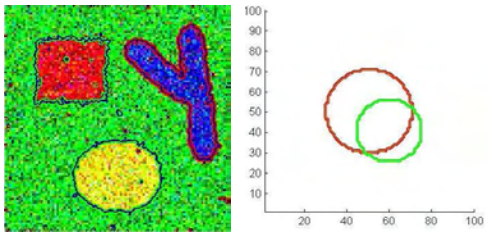


Figure 6: *First*: Result for Active contours. The resulting contours are still noisy. *Second*: Initial contours.

number of details, i.e., the number of regions, and leave only the "strongest" edges.

5 CONCLUSION AND FUTURE WORK

We generalized the existing single-channel MDL-based method to a multichannel one and applied it to color image partitioning and restoration. For synthetic color images, the results were compared to the ones obtained with Graph cuts, anisotropic diffusion, greedy region merging with energy, and active contours approaches. Our method produces best results in terms of color

Figure 7: Results with different *b* for the "woman" image. The initial image is the leftmost one. The parameter *b* is selected: 2, 5, 20 for the images from left to right, correspondingly.

restoration and boundary accuracy for our test cases. Moreover, the relaxation scheme with Jacobi iterations combined with a simple region merging procedure allows for fast computations due to straightforward parallelizability, which is important when applied to real-world problems.

We believe that this approach has potential in solving problems of image restoration. The future work directions include studying different noise models (e. g. correlated noise) for piecewise constant and piecewise smooth images as well as the models based on other MDL codes.

6 REFERENCES

[1] D. Baez-Lopez, F. H. Mendoza, and J. M. Ramirez. Noise in color digital images. *Circuits and Systems, Midwest Symposium on*, 0:403, 1998.

[2] A. Blake and A. Zisserman. *Visual Reconstruction*. MIT Press, 1987.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[4] Y. Boykov, O. Veksler, and R. Zabih. Efficient restoration of multicolor image with independent noise. Technical report, 1998.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

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

[7] F. Galland, N. Bertaux, and P. Refregier. Multi-component image segmentation in homogeneous regions based on description length minimization: Application to speckle, poisson and bernoulli noise. *Pattern Recognition*, 38(11):1926 – 1936, 2005.

[8] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.

[9] P. Grunwald. *A tutorial introduction to the minimum description length principle*. 2004.

[10] L. Guigues, J. Cocquerez, and H. Le Men. Scale-sets image analysis. *International Journal of Computer Vision*, 68:289–317, 2006.

[11] T. Ivanovska, L. Linsen, H. K. Hahn, and H. Voelzke. Gpu implementations of a relaxation scheme for image partitioning: Glsl vs. cuda. *Computing and Visualization in Science*, 14(5):217–226, 2012.

[12] T. Kanungo and B. Dom et al. A fast algorithm for mdl-based multi-band image segmentation. *Computer Vision and Pattern Recognition*, pages 609–616, 1994.

[13] I.B. Kerfoot and Y. Bresler. Theoretical analysis of multispectral image segmentation criteria. *Image Processing, IEEE Transactions*, 8(6):798–820, 1999.

[14] G. Koepfler, C. Lopez, and M. Morel. A multiscale algorithm for image segmentation by variational method. *SIAM Journal of Numerical Analysis*, 31:282–299, 1994.

[15] Y.G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3(1):73–102, 1989.

[16] T. C. M. Lee. A minimum description length-based image segmentation procedure, and its comparison with a cross-calidation-based segmentation procedure. *Journal of the American Statistical Association*, 95(449):259–270, 2000.

[17] T. C. M. Lee and T. Lee. Segmenting images corrupted by correlated noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:481–492, 1998.

[18] E. L. Lehmann and George Casella. *Theory of Point Estimation (Springer Texts in Statistics)*. Springer, 2003.

[19] S. Z. Li. *Markov random field modeling in image analysis*. Springer-Verlag New York, Inc., 2001.

[20] Q. Luo and T.M. Khoshgoftaar. Unsupervised multiscale color image segmentation based on mdl principle. *IEEE Transactions on Image Processing*, 15(9):2755–2761, 2006.

[21] V. Mezaris, I. Kompatsiaris, and M. G. Strintzis. Still image objective segmentation evaluation using ground truth. In *In Proceedings of the Fifth COST 276 Workshop on Information and Knowl-*

edge Management for Integrated Media Communication, 2003.

[22] http://vision.middlebury.edu/MRF/ Middlebury Vision.

[23] A. Mitiche and I. B. Ayed. *Variational and Level Set Methods in Image Segmentation*. Springer Berlin Heidelberg, 2011.

[24] D. Mumford and J. Shah. Boundary detection by minimizing functionals. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 1985.

[25] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):629–639, 1990.

[26] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.

[27] G. Sapiro and D. L. Ringach. Anisotropic diffusion of multivalued images with applications to color filtering. *Image Processing, IEEE Transactions on*, 5(11):1582–1586, 1996.

[28] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):1068–1080, 2008.

[29] L. A. Vese and T. F. Chan. A multiphase level set framework for image segmentation using the Mumford and Shah model. *International Journal of Computer Vision*, 50(3):271–293, December 2002.

[30] P. Villegas, X. Marichal, and A. Salcedo. Objective evaluation of segmentation masks in video sequences. In *In Proceedings of Workshop om Image Analysis for Multimedia Interactive Services*, 1999.

[31] S.C. Zhu and A.L. Yuille. Region competition: unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *IEEE Trans. on PAMI*, 1996.

A Study on the Animations of Swaying and Breaking Trees based on a Particle-based Simulation

Yasuhiro AKAGI
Tokyo University of Agriculture and Technology
Naka-cho 2-24-16
184-8588, Koganei, Tokyo
scout@cc.tuat.ac.jp

Katsuhiro KITAJIMA
Tokyo University of Agriculture and Technology
Naka-cho 2-24-16
184-8588, Koganei, Tokyo
Kitajima@cc.tuat.ac.jp

ABSTRACT

In this paper, we propose a particle-based simulation method to create the animations of swaying and breaking trees. Since the shapes of trees and their realistic motions are usually complex, it is difficult for us to create an animation of a swaying tree manually. Therefore, to produce films and image contents which contain a natural scene of swaying and breaking trees, it takes a lot of work to create the animation. To solve this problem, it is important that how to calculate interactions between a tree and wind to automatically generate the swaying and breaking motions of a tree by using a physical simulation. We model both a tree and wind as particles to simulate the interactions. The advantage of the particle-based method is that the method is robust for changing of the topology of wind and the branching structure of a tree. Our results show that the proposed method can naturally represent the breaking behavior of a tree and the wind flow around the tree by using the particle-based simulation of the wind.

Keywords

Tree animation, Breaking branches, Fallen leaves, Particle-based simulation, SPH method, Elastic deformation

1. INTRODUCTION

The 3D modeling and creation of animations of natural scenes is one of the most time-consuming works of producing films and videos[Chn11]. It is difficult to manually create an animation of swaying trees that has realistic movements of branches and leaves in an environment having a complex wind flow. Therefore, there are many studies on the animation of swaying trees on the basis of tree dynamics and wind simulations. We propose a practical method of generating tree animations that simulate the swaying motions and breaking behaviors of trees by using a particle-based model. We represent both the trees and wind by particles and links to be able to simulate the breaking behaviors of trees. The most advantage of the method using the particles is that the simulation method can freely divide the tree structure into broken branches and fallen leaves. This method also represents the change of the movement which is caused by the breaking and changing the shape of the tree.

Previous work

Most of these studies propose a method of simulating the dynamics of branches by using beam structures[AWZ09]. These methods simulate the bending motions of the beam structures (branches) on the basis of external forces (wind). For instance, Habel [HKW09] and Yang[YHY+10] proposed a method of automatically animating swaying trees using beam structures. Diener [DRB+09] proposed a simpler method of simulating bending branches. This method divides a tree into the sections and simulates the movements of branches only by the rotation of these sections. The method enables us to generate tree animations that contain thousands of trees in real time. Then, our method simulates the deformation of the beam structure of a branch by using the particles and connection of the particles.

In recent years, there are several studies on the simulation of the breaking behaviors of branches and falling leaves for generating tree animations [BCN+10]. Saleem[SCZ+07] proposed a method that simulates the breaking behaviors of branches. In this method, the breaking positions of a branch are decided on the basis of the threshold velocity, which is the velocity of the wind around the branch. When the breaking position of a branch is detected, the method detaches the broken branch from the tree and simulates the falling motion of the branch. Since the threshold velocities are set by the users beforehand, it takes a considerable amount of time to create the tree animations[Chn11]. Moreover, this method cannot

detect the broken positions of branches and cannot generate the cross-sectional shape of a broken branch automatically.

On the other hand, to generate fluid animations, many studies on the simulation of complex movements such as broken waves, splashing, and pouring have been conducted [AT11][CBP05]. These studies are based on the SPH method[Luc77], which is a particle-based fluid dynamics method. One of the features of the SPH method is that it represents a fluid as a set of particles that makes it easy to simulate the separation and fusion of the fluid. Another feature of the SPH method is that it is easy to parallelize. With the use of this method, Goswami[GSS+10] succeeded in accelerating the generation of fluid animations by using GPUs. Stava et.al.[SBB+08] proposed "Surface particles" to simulate the erosion of a terrain by defining the relationship between particles and terrain.

Moreover, there are several studies based on the particle method for simulating the deformations of elastic objects. Muller[MC11] proposed a particle-based simulation method for elastic objects. One of the features of this method is that it effectively represents elastic objects by using ellipsoid particles. Gerszewski[GBB09] also succeeded in deforming elastic objects robustly by using particle-based models. Since the particle-based approach has the advantage of geometric changes, there are several studies on the destruction simulation of solid objects. Pauly[PKA+05] succeeded in simulating the destruction of solid objects by replacing the finite element meshes of an object with particles. [WRK+10] proposed a method for simulating deformation and fracturing of elastic objects. This method efficiently simulates the elastic objects with remeshing technique. Since the trees have complexly branched structures and intersections, we use the particle-based method to simulate the deformation and fracturing of trees to prevent the miss generation of meshes.

2. PHYSICAL SIMULATION OF SWAYING AND BREAKING TREES

In this section, we propose a simulation method to create an animation of swaying and breaking trees. This method is based on a particle-based simulation of wind and elastic deformations. We model both wind and trees as particles that have the same physical parameters in order to handle the interactions between the wind and the trees independently. The advantages of using the particle-based method are as follows:

1. If the physical parameters of the particles, such as position and velocity, are discrete, the interactions between the particles can be simply simulated as a particle-to-particle problem.
2. The particle-based structure has an advantage with respect to a breaking behavior that contains a topology change.
3. As the particles are discrete, it is easy to speed-up the simulation by using a parallel processing technique.

	Parameter	Symbol
Particle Prt_i	Position	x_i
	Velocity	u_i
	Acceleration	a_i
	Interaction Radius	r_i
	Mass	m_i

Table 1. Physical parameters of a particle

In the following sections, we describe the equations of our simulation by using the symbols given in Table 1.

Types of interaction models

When we perform a particle-based physical simulation, it is important to define the interactions between particles. In order to distinguish between the physical characteristics and interactions of the wind and the trees, we define the following four types of particles:

Wind : wind particle

Branch: particle for a part of a branch

Leaf : leaf particle

Bone : particle for the bone structure of a tree

We define three of the abovementioned types of particles for representing a tree. Two of these three are the types that help distinguish between the branches and the leaves. The fourth one represents the structure of a tree, such as the bone structure for a CG character. This bone particle is used for calculating the restoring force. We will describe the effect of this type of particles in Section 2.

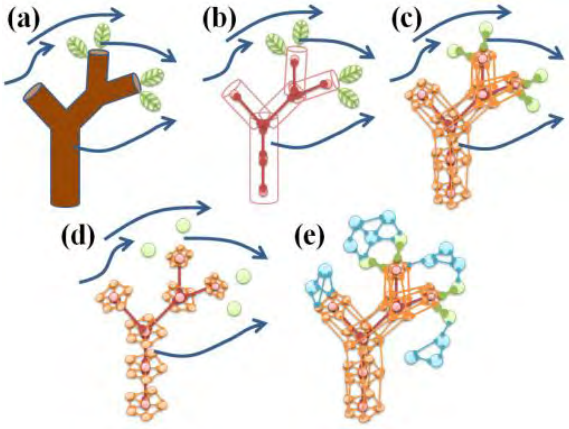


Figure 1. Generation process of the particle and link-structure based tree model form a polygon-based tree model

Particle-based tree modeling

In this section, we describe a method of modeling trees on the basis of the abovementioned four types of particles. To simulate the swaying motions and breaking behaviors of a tree, we define a link structure of the particles that are used for replacing the 3D tree model. The link structure represents the relationships between the particles for detecting an interaction force between two particles. In the proposed method, we require two types of information for the 3D tree model. One is the information regarding the differences between branches and leaves, and the other is the information regarding the tree structure and the positions of the branching points. By using the information of the tree, the proposed method generates the particles and link structures by following steps:

		Destination			
		Wind	Branch	Leaf	Bone
Source	Wind	f_{wind}	f_{wind}	f_{wind}	f_{wind}
	Branch	f_{wind}	f_{spr}	f_{spr}	f_{spr}
	Leaf	f_{wind}	f_{spr}	-	-
	Bone	f_{wind}	$f_{spr} + f_{shr}$	-	f_{bnd}

Table 2. Interaction model

(a) Define central axis

We use the links that are named the "central axis" [TMW02]. The central axis connects two particles to form the bone structure of a tree. We generate these links on the basis of the information regarding the positions of the branching points of the 3D tree model. Then, the proposed method generates subdivision particles at regular intervals in order to calculate the bending of a branch.

(b) Replace branches and leaves with particles

The particles for representing branches are generated on the circumference of a circle whose center is formed by the particles of the central axis generated in step (a). The radius of the circle is the same as that of the branch, and the normal vector of the circle is the same as the direction of the central axis. The particles for representing the branches are generated on the central position of the leaf.

(c) Generate link structure

We generate the link structure of a tree on the basis of the particles that are generated in the previous step. Particles for various parts of a branch are linked side by side on the circumference of the circle and are linked to the particles of the central axis, which are the center particles of the circle. These particles are also linked to particles that are linked to the neighboring circles. The leaf particles are linked to the nearest particles of the parent branch.

(d) Define particles and links for wind

We define the cubic area for simulating the wind, and the wind particles are generated in this area at regular intervals. The links from a wind particle are generated on the basis of the influence radius of the particle, and the links are updated in each time step of the simulation.

Next, we define the four types of interaction forces as follows:

f_{wind} : Force of the wind from the fluid simulation
f_{spr} : Internal force of a tree based on a mass-spring system
f_{bnd} : Restoring force from the central axis
f_{shr} : Shear stress from the structure of a branch

Table 2 shows the relationships between the four types of particles and the four types of interaction forces. The names of the types in the first column are the source of the interaction force, and those in the second row from the top are the destination.

The wind particles affect all types of interactions. We calculate the interaction force f_{wind} using the Navier-Stokes equations. Further, we simulate the elastic deformations of a tree by using three types of interaction forces. We will describe the method for simulating the elastic deformations in fifth Section.

Wind simulations

The interactions from the wind particles are calculated using the Navier-Stokes equations, which are one of the fundamental equations in the field of fluid dynamics. We apply the SPH method, which is a numerical solution of fluid dynamics, to simulate the airflow around a tree. Since the SPH method handles both fluids and objects as particles, it can be used for simulating the interactions between the wind and the trees. Equation (1) gives the interaction force $f_{wind\,i}$, which is the force exerted by all types of particles on particle Prt_i.

$$f_{wind\,i} = \mu\nabla^2 u_i - \nabla p_i + \rho_i f \quad (1)$$

μ: viscosity, p_i: pressure,

ρ_i: density, f: external force

The first factor on the right side of equation 1 is the viscosity factor and the second factor is the pressure factor. The following equations are the differential equations for each factor.

$$\mu\nabla^2 u_i = \mu \sum_j m_j \frac{u_j - u_i}{\rho_j} \nabla W(x_j - x_i) \quad (2)$$

$$\nabla p_i = \sum_j m_j \frac{p_j + p_i}{2\rho_j} \nabla W(x_j - x_i) \quad (3)$$

The variable W denotes the kernel function of the SPH method[MC11] between Prt_i and Prt_j.

By using these equations, we simulate the interactions between the wind and the trees on the basis of the particle definitions. Then all of the relationships between wind and trees are detected based on the radius of the particles. In this process, since we deal with the particles of trees as same as the particles of wind, the two-way effect are calculated by the SPH method. When we implement the SPH method, we set the natural density of the air for the parameter ρ_i and the mass of the tree m_j is calculated with $0.5g/cm^3$.

Interaction models for branches and leaves

We described the link structures of branches and leaves for simulating elastic deformations of a tree in the second section. In this section, we describe the three types of interaction forces that affect the particles for representing a tree. Figure 2 shows the link structure and the interaction forces of a branch.

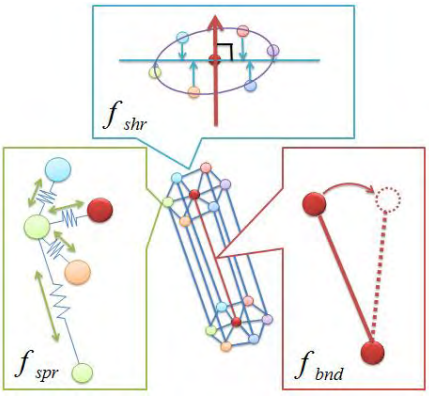


Figure 2. Three types of interaction forces for simulation of the elastic deformation of a branch

The first interaction force $f_{spr\,i}$ is the internal force of a tree based on a mass-spring system. The value of $f_{spr\,i}$ is given by equation 4.

$$f_{spr\,i} = \sum_j k_{i.j}\left(\left(x_j - x_i\right) - L_{i,j}\right) \quad (4)$$

$k_{i.j}$: Spring constant between Prt_i and Prt_j

$L_{i,j}$: Initial length between Prt_i and Prt_j

The second interaction force $f_{bnd\,i}$ is the restoring force from the central axis. Now, there are two particles for representing the central axis, Prt_i (top) and Prt_j (bottom). Then, we define the direction vector of the central axis $V_{center\,i}\left(= x_i - x_j\right)$. When we determine the restoring force, we require the relative rotation angle of the central axis to be against the initial rotation. To determine the rotation angle, we calculate the quaternion Q_j, which represents the rotation from $V_{center\,j}$ to $V_{center\,i}$ on the basis of the initial position of the tree beforehand.

$$Q_j(V_{center\,j}) = V_{center\,i} \quad (5)$$

Then, we define the interaction force $f_{bnd\,i}$ as follows:

$$f_{bnd\,i} = G_{b\,i}\left(Q_j\left(L_{i,j}|V_{center\,j}|\right) - V_{center\,i}\right) \quad (6)$$

$G_{b\,i}$: Elastic modulus

The third interaction force f_{shr} is the shear stress from the structure of a branch. This force moves Prt_i for representing branches on the circle whose center is formed by the particles of the central axis Prt_j. The normal vector of the circle is given by equation 7:

$$V_{normal\,i} = \frac{V_{center\,i}}{|V_{center\,i}|} \quad (7)$$

By using the normal vector, we can determine the distance and the direction from Prt_i to the circle. Equation 8 gives the relationship between both the distance and the direction and the interaction force f_{shr}.

$$f_{shr\,i} = G_{s\,i}\left(V_{normal\,i} \cdot x_j - V_{normal\,i} \cdot x_i\right)V_{normal\,i} \quad (8)$$

$G_{s\,i}$: Modulus of rigidity

We simulate the elastic deformations of trees by using these three types of interaction forces.

Breaking conditions of a tree

In this section, we describe the determination of the broken locations of a tree. We represent the breaking behavior of a tree by erasing the links between the particles that compose the particle-based model of the tree. We assign the limitation length of a stretch $e_{i,j}$ and a compress $c_{i,j}$ to each link of the tree model. Usually the breaking conditions are detected based on the stress of an object. In our method, the stress of a link can be calculated with $f_{spr\,i}$. Since the value of $f_{spr\,i}$ simply proportion the length between particles, we use the length to detect the broken position to be easy to set the threshold parameters by users. To determine the broken link between Prt_i and Prt_j, we use the strain measure of the link $d_{i,j}$.

$$d_{i,j} = \left|\frac{(x_j - x_i)}{L_{i,j}}\right| \quad (9)$$

If the strain rate of the link $d_{i,j}$ is greater than $e_{i,j}$ or lesser than $c_{i,j}$, the link is erased from the tree model.

$$IsBroken = \begin{cases} true & d_{i,j} \geq e_{i,j} \ \ or \ \ d_{i,j} \leq c_{i,j} \\ false & e_{i,j} > d_{i,j} > c_{i,j} \end{cases} \quad (10)$$

The threshold parameters $e_{i,j}$ and $c_{i,j}$ are given by the users. Then, if half of the links that are generated along the central axis are erased, the link of the central axis is erased.

3. GENERATION OF BROKEN BRANCH

In this section, we describe a method of modeling a broken shape of a branch. This method automatically generates a polygon model that represents a cross-sectional shape of a broken branch on the basis of the link structure of a tree described in Section 2.

Internal model of a branch

A branch is constructed by a bundle of fibers, and the cross-sectional shape of a broken branch is uneven. We define the internal model of a branch that represents the difference in the density, radius, and color of fibers in order to generate the unevenness. This internal model consists of three layers: core, wood, and bark. By controlling the thickness of each layer, this model can handle a branch that has an internal cavity, such as a bamboo.

Polygon model for broken branch

In this section, we describe the generation of a polygon model for a broken branch on the basis of the internal model of a branch (Figure 3). The generation procedure is as follows:

(a) The layered structure of a branch.

(b) The vertices that represent the fibers are placed randomly into the area of each layer according to its density.

(c) By using the Delaunay triangulation, we connect the vertices to form triangles.

(d) The triangles generated in step (c) are expanded to form triangle poles.

In our implementation, the broken shape is dynamically generated when the branch is broken. In our preliminary experiment, the computation time of the generation process is match faster than other simulations (< 0.005sec.).

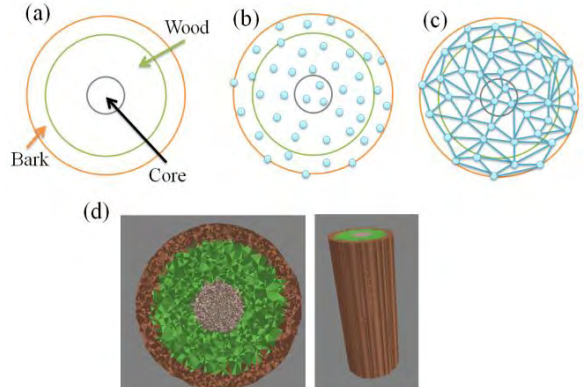


Figure 3. Generating process of a polygon model for broken branch.

Cross-section of a broken branch

In this section, we describe the division of the polygon model generated in Section 3. When a natural branch is broken at an arbitrary position on the surface of the branch, the breaking process proceeds to the other side. Then, the cross-sectional shape of the branch forms a curve that goes down to the bottom gradually. Therefore, we define the side curve of the cross-sectional shape on the basis of equation 11. This equation gives the drop rate F of the side curve (Figure 4 upper-side).

$$F(x) = L * \alpha \left(\frac{(C-x_0)\cdot(x-x_0)+1}{2r^2}\right)^2 \qquad (11)$$

L: Length of central axis, α: Brittleness,

C: Position of central axis, x_0: Starting position of the breaking.

The bottom side of the figure 4 shows the cross-sectional shape of the branch.

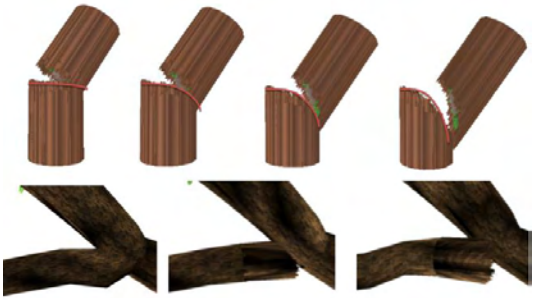


Figure 4. Cross-sectional shapes of a branch.
Upper-side: Braking goes down to the bottom.
Bottom-side: Braking behavior in a scene.

4. Results and Discussions

Particle-based tree modeling

We generate a particle- and link-based tree model by using the method described in Section 2. Figure 5 shows the polygon-based model and the particle-based model of a tree. One of the advantages of this method is that since the intervals between the particles can be changed, the users can freely change the details of the simulation of the tree.

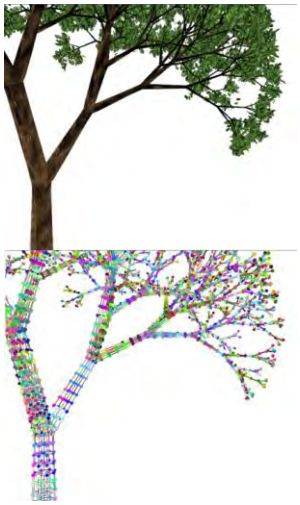


Figure 5. Result of generating particle-based model from a 3D model of a tree.

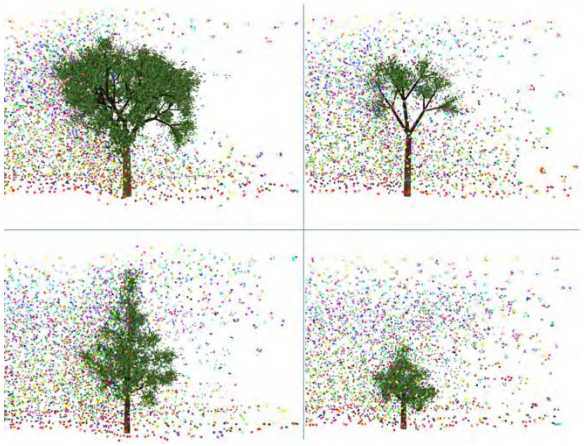


Figure 6. Comparison of the interaction between a tree and wind flow according to the differences in the shapes of trees.

Interactions between wind and tree

In this section, we present the results of the tree animations that contain the breaking behaviors of branches and falling leaves.

4.1.1 Swaying branches and leaves

Figure 6 shows the comparison of the wind flow according to the differences in the shapes of trees. The color dots on the images are the particles used for representing the wind. Since the branches and leaves of the trees block the wind particles, the larger and denser tree the more wind flows around the tree. These results show that the proposed method can represent the natural interactions between the tree and the wind.

4.1.2 Breaking branches and falling leaves

Figure 7 shows the Comparison of the broken conditions of branches according to the differences in the velocity of the wind. The red circles are the signs of the broken branches. In the previous study, Saleem et.al. [SCZ+07] enable to visualize the flying effect of the branches. However, this method calculates the broken conditions by using the angle of the branch and velocity of wind. Therefore the Salem's method could not visualize the difference of the broken position of a tree according with the velocity of wind. In our result, the image on the top of figure 7 is the result of the lowest velocity of the wind. This results show that there are any broken branches and a few fallen leaves. The faster the velocity of the wind causes the more number of the broken branches and fallen leaves. This result shows that the proposed method can visualize the difference of the broken position according with the velocity of the wind and the structure of the tree.

4.1.3 Implementation and performance

In this section we discuss the simulation performance of our method. We measured the processing time on a 2.66Ghz Xeon PC with 8GB RAM. We set the radius of the particles for wind 0.05m and place the particles for the trees every 0.1m. The time-step of our simulation is 0.01sec. Table3 shows the results to compare the time scaling according to the number of the particles for trees. This result shows that the increase of the number of the particles causes the increase of both the time for simulating wind and the tree. Since our method calculates the relationship between a tree and wind in the process of SPH method, the processing time for SPH method also increase according to the particles of a tree. The table 4 shows the results to compare the time scaling according to the number of the particles for wind. This result shows that since the relationship between the tree and wind are considered in the process of SPH method, the processing time for computing the tree deformation is not changed according to the number of the particle of wind. Since the processing times are not real-time,

Number of particles		Calculation time per fame (sec.)		
Wind	Tree	SPH	Tree Deformation	Total
7680	16877	0.225	0.134	0.360
	22319	0.356	0.178	0.535
	31812	0.565	0.264	0.829
	44222	0.958	0.367	1.325
	72759	2.412	0.617	3.028
	102282	4.350	0.869	5.219

Table 3. The time scaling according to the number of the particles for trees.

Number of particles		Calculation time per fame (sec.)		
Wind	Tree	SPH	Tree Deformation	Total
2560	21900	0.262	0.177	0.439
5120		0.310	0.174	0.484
9984		0.395	0.174	0.569
17664		0.610	0.176	0.787
25088		1.096	0.175	1.270
35072		2.312	0.181	2.492

Table 4. The time scaling according to the number of the particles for wind.

5. CONCLUSIONS

We proposed the particle-based simulation method to create the animations of swaying and breaking trees. We obtained the following results:

(1) We defined the four types of particles (wind, branch, leaf and bone) and the four types of interaction forces (wind, internal force, restoring

force and shear stress) between the particles. By using the models, we can simulate the swaying and breaking behaviors of trees.

(2) We proposed the generation method of a polygon model for a broken branch. This method is able to generate the cross-sectional shape of a broken branch on the basis of the link structure of a tree.

(3) Our results show that the animation of swaying and breaking trees is automatically generated by the proposed method. By using the particle-based simulation of the wind, the proposed method can represent that the wind flow naturally around the tree.

In a future work, we will accelerate our simulation method to be able to generate animations in real-time by using parallel processing on GPU.

6. ACKNOWLEDGMENTS

This research was supported by Adaptable and Seamless Technology Transfer Program through Target-driven R&D (A-Step), Japan Science and Technology Agency.

7. REFERENCES

[AT11]Ando, R., Tsuruno, R., A particle-based method for preserving fluid sheets, In Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '11), pp.7-16, 2011.

[AWZ09]Ao, X., Wu, Y., Zhou, M., Real time animation of trees based on BBSC in computer games, International Journal of Computer Games Technology, Article 5, 2009.

[BCN+10]Baxter, R., Crumley, Z., Neeser, R., Gain, J., 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), pp.101-110, 2010.

[CBP05]Clavet, S., Beaudoin, P., Poulin, P., Particle-based viscoelastic fluid simulation, In Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '05), pp.219-228, 2005.

[Chn11]Chng. E., Realistic Placement of Plants for Virtual Environments. IEEE Comput. Graph. Appl. 31, 4, pp.66-77, 2011.

[DRB+09]Diener, J., Rodriguez, M., Baboud, L., Reveret, L., Wind projection basis for real-time animation of trees, Computer Graphics Forum (Proceedings EUROGRAPHICS 2009), 28, 2, pp. 533-540, 2009.

[GSS+10]Goswami, P., Schlegel, P., Solenthaler, B., Pajarola, R., Interactive SPH simulation and rendering on the GPU, In Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10), pp.55-64, 2010.

[GBB09]Gerszewski, D., Bhattacharya, H., Bargteil, W.A., A point-based method for animating elastoplastic solids, In Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '09), pp.133-138, 2009.

[HKW09]Habel, R., Kusternig, A., Wimmer, M., Physically Guided Animation of Trees, Computer Graphics Forum (Proceedings EUROGRAPHICS 2009), 28, 2, pp.523-532, 2009.

[Luc77]Lucy, L., A numerical approach to the testing of the fission hypothesis, 82, pp.1013-1024, 1977.

[MC11]Muller, M., Chentanez, N., Solid simulation with oriented particles, ACM Trans. Graph., 30, 4, Article 92, 2011.

[PKA+05]Pauly, M., Keiser, R., Adams, B., Dutr, P., Gross, M., Guibas, J.L., Meshless animation of fracturing solids, ACM Trans. Graph. 24, 3, pp.957-964, 2005.

[SBB+08]Stava, O., Benes, B., Brisbin, M., Krivánek, J., Interactive Terrain Modeling Using Hydraulic Erosion. In Proceedings of Symposium on Computer Animation. pp.201-210, 2008.

[SCZ+07]Saleem, K., Chen, S., Zhang, K., Animating Tree Branch Breaking and Flying Effects for a 3D Interactive Visualization System for Hurricanes and Storm Surge Flooding, In Proceedings of the Ninth IEEE International Symposium on Multimedia Workshops (ISMW '07), pp.335-341, 2007.

[TMW02]Tobler, F. R., Maierhofer, S., Wilkie, A., A multiresolution mesh generation approach for procedural definition of complex geometry. In Shape Modeling International 2002, 11, 15, pp.35-43, 2002.

[WRK+10]Wicke, M., Ritchie, D., Klingner,M. B., Burke, S., Shewchuk, R. J., O'Brien, F. J, Dynamic Local Remeshing for Elastoplastic Simulation, In Proceedings of ACM SIGGRAPH 2010, pp. 49:1–11, 2010.

[YHY+10]Yang, M., Huang, M., Yang, G., Wu, E., Physically-based animation for realistic interactions between tree branches and raindrops, In Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology (VRST '10), pp.83-86, 2010.

Figure 7. Comparison of the broken conditions of branches according to the differences in the velocity of the wind.

A Statistical Framework for Estimation of Cell Migration Velocity

Guangming Huang Jiwoong Kim Xinyu Huang Gaolin Zheng Alade Tokuta
Department of Mathematics and Computer Science
North Carolina Central University
1801 Fayetteville Street
Durham, NC 27707, USA
{ghuang, jkim7, huangx, gzheng, atokuta}@nccu.edu

ABSTRACT

Migration velocity of cell populations *in vitro* is one of important measurements of cell behaviors. As there are massive amount of cells in one image that share similar characteristics and are highly deformable, it is often computational expensive to track every individual cell. It is also difficult to track cells over a long period of time due to propagation of segmentation and tracking errors. This paper presents an algorithm to estimate migration velocity of cell populations observed by time-lapse microscopy. Instead of tracking cells individually, our proposed algorithm computes mutual information between image blocks of consecutive frames. The migration velocity is then estimated by a linear regression, with mutual information and foreground area ratio as input. Experiments on a variety of image sequences verified that our algorithm can give accurate and robust estimation under different situations in real-time.

Keywords

Cell Migration, Mutual Information, Linear Regression.

1. INTRODUCTION

It is important to measure cell migration velocity in many biomedical applications, such as wound healing assay of cell monolayers [YPW+04] and analysis of red blood cell in microcirculation [WZH+09]. For cell populations, there are mainly two obstacles to estimate the migration velocity accurately and robustly. First, all the cells in a population have very similar characteristic, such as shape and intensity. Second, cells are often highly deformable. For example, two cells could merge into one cell, and one cell could divide to two or more cells. As a result, it could be difficult and computational expensive to track every cell in image sequences. Figure 1 shows three examples of cell populations. Cells in some types of images even have very similar intensities to the background as shown in figure 1(b). Thus, the segmentation algorithms based on intensity values would easily fail.

In this paper, we present an efficient and novel algorithm to estimate cell migration velocity. Our algorithm first computes mutual information and foreground ratio between image blocks of two consecutive frames. A linear regressor is then trained and applied to estimate migration velocity.

Mutual information has been widely used to align two images in many medical applications [PMV03] in order to reduce the error during the image acquisition (*e.g.,* finger jigging). However, to our knowledge, there is no work that uses mutual information as an input variable of regression for the estimation of cell migration velocity. As there are no individual cells involved in the estimation process, our algorithm contains no accumulated segmentation and tracking errors.

Therefore, the proposed algorithm has several advantages. First, accurate segmentation and data association of cell contours are not required. Second, it can be performed in real-time without using motion trackers for all the cells. Third, since tracking accuracy is not an issue (*e.g.,* there are no accumulation errors), it can be used to estimate migration velocity for a long period.

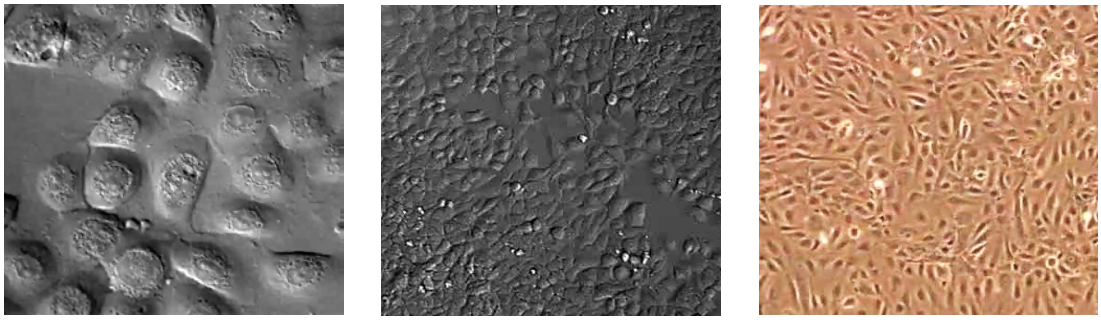


Figure 1. Examples of microcopy images of cell populations: (*left*) primary keratinocytes [Cel09]. (*middle*) cancer cells captured spinning disc confocal microscope [Mar09]. (*right*) human umbilical vein endothelial cells (HUVEC) [Yam09].

The remainder of this paper is organized as follows. Section 2 describes related work. Our proposed algorithm is given in section 3. Section 4 shows the experiments using three different datasets. The conclusion is given in section 5.

2. RELATED WORK

Wu *et al.* proposed an approach to measure velocity of red blood cell from capillary video using the optical flow technique [WZH+09]. In order to apply the optical flow technique, the skeleton of vessel needs to be extracted first based on a set of pre-processing steps, such as connected component labeling, thinning, and length pruning. These pre-processing processes may not be applied to other types of cells in general due to occlusions, deformations, and even varying illuminations. More importantly, the velocity determination in this approach is based on two assumptions: a) intensity of each cell does not change over time; b) the surrounding area of the cell move in a similar manner. These two assumptions are fundamental to apply the optical flow on the skeletons. They however, are too restricted and cannot be extended in general. Other approaches [DSA+08, LGM04] that based on detection and graph extraction of vessel shapes also have similar problems.

In [LMC+08], Li *et al.* proposed an automated tracking algorithm to track hundreds to thousands of cells and construct lineages simultaneously. This tracking system first segments candidate cell regions and tracks them over frames by forming a minimization problem with a topological constraint. Then this system predicts and filters the cell motion dynamics using interacting multiple model filters, and construct lineages by checking the entire tracking history. The proposed system can be used to analyze a number of cell behaviors including migration, division, death, and so on.

According to [MDI+09], tracking in cell can be divided into two stages, segmenting individual cells and connecting cells over time. However, since each possible candidate cell needs to be considered, the whole process could be computational expensive. For the purpose of estimating migration velocity, the algorithms based on tracking individual cells could be complicated and hence may not be the best choice. Moreover, common used segmentation algorithms based on intensity values could easily fail to distinguish between background areas and candidate cells.

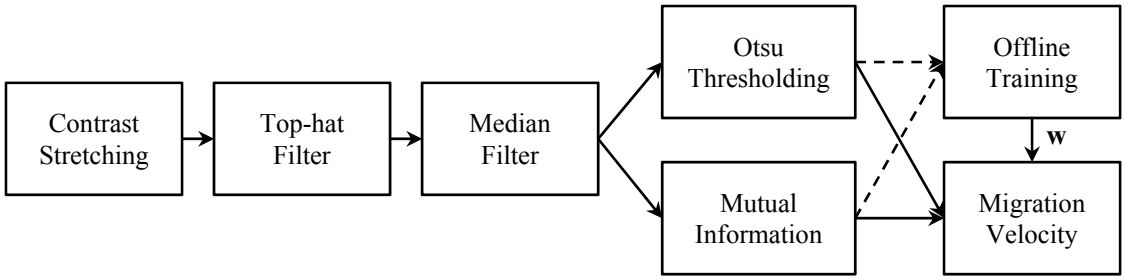


Figure 2. Overview of our methodology

3. METHODOLOGY

Our algorithm can be divided into three modules, 1) image enhancement and foreground detection; 2) computation of mutual information between image blocks; 3) velocity estimation using linear regression. Figure 2 gives an overview of the algorithm.

3.1 Image Enhancement and Foreground Detection

The purpose of image enhancement is to reduce noise and enhance the image contrast. For simplicity, we assume two image frames has been aligned. This can be achieved by an affine transformation based on an estimated homography [HZ04], or a non-linear transformation using mutual information [PMV03].

First, we enhance the image by a linear contrast stretch that enlarges the range of intensity values to the entire available range. If the cells appear darker than the background, the images are inverted so that the cell interior areas are brighter. Next, the enhanced image is convolved with a 7×7 top-hat filter, which is often used to detect bright features on a dark background. A 3×3 median filter is then applied twice to remove small grey objects that could be noise or artifacts caused by the top-hat filter. The output image after the median filter is the input image for the computation of mutual information. We further detect the foreground by using Otsu thresholding [Otu79] to obtain a binary image where the white pixels indicate the foreground. Figure 3 visualizes these image processing steps.

Unlike many algorithms that require fairly accurate foreground and cell detections, we only need to detect a rough foreground as shown in Figure 3(f). Moreover, there are no accumulated segmentation errors.

3.2 Computation of Mutual Information

Mutual information is usually used to test the independence between two random variables **x** and **y**. If two random variables are independent, the joint probability $p(\mathbf{x}, \mathbf{y})$ can be factorized into the product of their marginals $p(\mathbf{x})p(\mathbf{y})$. In our application, the random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are two same image blocks from two consecutive image frames.

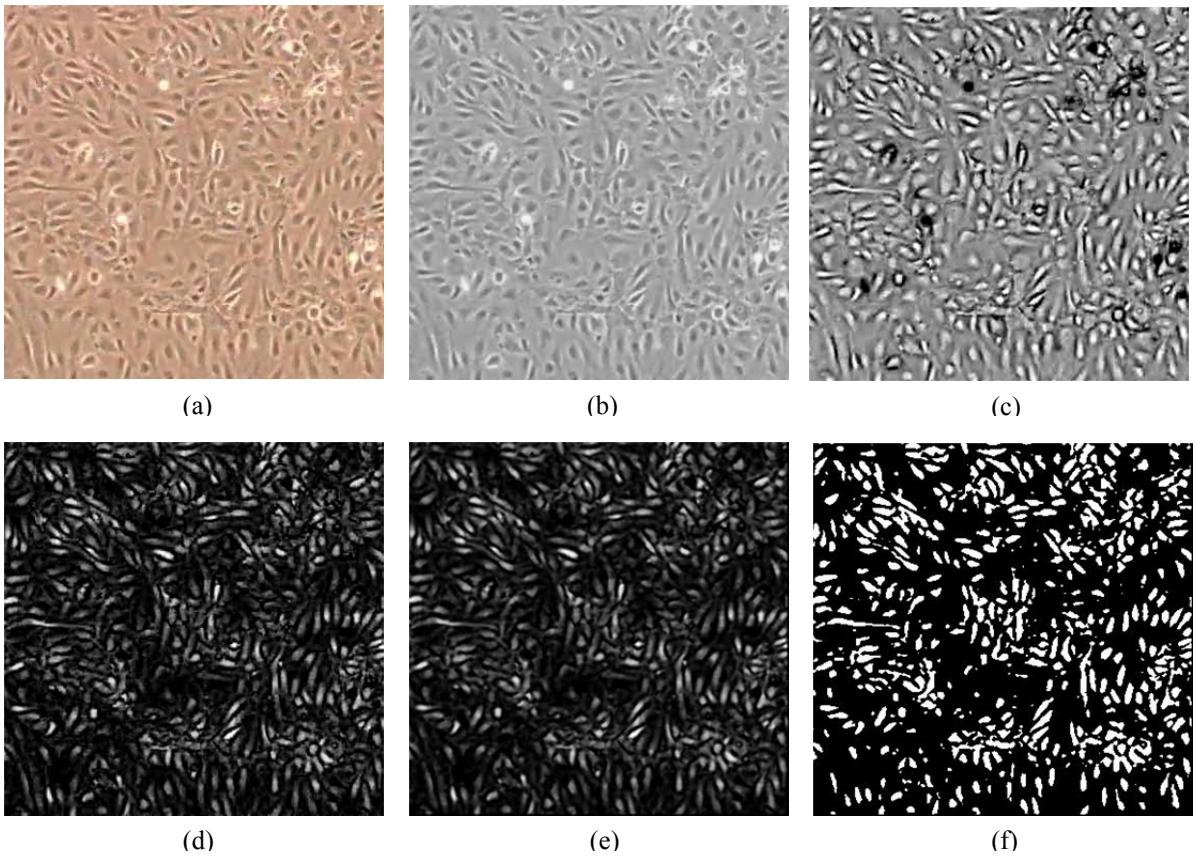


Figure 3. Image enhancement and foreground detection. (a) Original color image. (b) Grayscale image. (c) Result after intensity inversion and contrast stretching. (d) Result of top-hat filtering. (e) Output of median filtering. (f) Foreground detection by Otsu thresholding.

We do not use the entire image frames as the random variables. This is mainly because that the majority of an image could be the background, in which case, when the entire images are used, the mutual information between them could mainly reflect the differences between backgrounds. Furthermore, when the captured image has a high resolution, it also could be inefficient since the memory storage is increased. Therefore, we divide each frame into a set of image blocks with overlapping areas. The size of the image block is determined by the maximum of cell migration velocity, which can be easily estimated by the visual inspection. For many types of cells, the size of an image block is often around a few times that of a single cell.

Let us denote an image block as **x** and the same image block in the next image frame as **y**. The mutual information between the variables **x** and **y** is defined as

$$I(\mathbf{x},\mathbf{y}) = \mathrm{KL}(p(\boldsymbol{x},\boldsymbol{y})||p(\mathbf{x})p(\mathbf{y})) = \sum_{i=1}^{N}\sum_{j=1}^{N} p(x_i,y_j)\ln\left(\frac{p(x_i,y_j)}{p(x_i)p(y_j)}\right)$$

where KL(·) is known as the Kullback-Leibler divergence, and $p(\mathbf{x})$ and $p(\mathbf{y})$ are histogram distributions of two image blocks. We can see that $I(\mathbf{x},\mathbf{y}) = 0$ if and only if two image blocks are independent, which indicates a very large migration. The larger the $I(\mathbf{x},\mathbf{y})$, the more similar two image blocks are, which indicate a small migration.

3.3 Velocity Estimation using Linear Regression

For each pair of image blocks **x** and **y**, we describe the cell migration using two features: the mutual information *I* and the difference between foregrounds of **x** and **y**. The mutual information *I* measures independence between two image blocks including both foreground and background. The difference between two foregrounds is measured by the ratio of non-overlapping foreground to the union of the two foregrounds, which is defined by

$$D = \frac{|\mathbf{x}_F \cup \mathbf{y}_F - \mathbf{x}_F \cap \mathbf{y}_F|}{|\mathbf{x}_F \cup \mathbf{y}_F|}$$

where subscript *F* indicates the foreground. In general, the smaller the ratio, the more similar two image blocks are. Thus, the inputs for the regression are the mutual information *I* and area ratio *D*, and the output is the cell migration velocity *v*. Since there could exist multiple moving cells in one image block, the maximum velocity is chosen as output.

In order to avoid over-fitting problem, we only consider the second order of the inputs. Therefore, the possible variables include I, D, I^2, D^2, and ID. We adopt the *forward selection* to select a suitable model for the data. In this model selection approach, we add one variable that results in the largest reduction in sum squared errors (SSE), and then carry out a hypothesis test to determine whether this reduction in SSE is significant. If the reduction is significant, we continue the adding process and stop otherwise. The results show that the full model is the most suitable regression model when the area ratio can be robustly estimated. The regression model is given by

$$v(I,D,\mathbf{w}) = [1 \;\; I \;\; D \;\; I^2 \;\; D^2 \;\; ID]^T\mathbf{w}$$

where $\mathbf{w} = (w_0,\dots,w_5)^T$.

In order to estimate the parameters **w**, we first developed an interactive user interface to measure the velocities of a set of sample cells ($N \approx 30$) by manually clicking centroids of the same cells in two image frames. Then the parameters **w** can be estimated by the normal equations

$$\mathbf{w} = (\boldsymbol{\phi}^T\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^T\boldsymbol{v}$$

where $\boldsymbol{\phi}$ is $N \times 6$ design matrix given by

$$\boldsymbol{\phi} = \begin{pmatrix} 1 & I_1 & D_1 & I_1^2 & D_1^2 & I_1D_1 \\ 1 & I_2 & D_2 & I_2^2 & D_2^2 & I_2D_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & I_N & D_N & I_N^2 & D_N^2 & I_ND_N \end{pmatrix}$$

It could be very difficult to estimate the area ratio *D* for some types of microcopy images. For example, for the image type shown in figure 1(b), the intensities of both foreground and background are very similar and illumination conditions also change when cells are moving. Therefore, the most exiting cell segmentation algorithms based on intensity values would fail to detect foreground and moving cells accurately.

For these types of images, we discard the area ratio *D* in our regression model and only use the mutual information. Thus, the model could be changed to a polynomial regression with order 3, which is given by

$$v(I,\mathbf{w}) = [1 \;\; I \;\; I^2 \;\; I^3]^T\mathbf{w}$$

where $\mathbf{w} = (w_0,\dots,w_3)^T$. The normal equations remain same and the $N \times 4$ design matrix is defined by

$$\boldsymbol{\phi} = \begin{pmatrix} 1 & I_1 & I_1^2 & I_1^3 \\ 1 & I_2 & I_2^2 & I_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & I_N & I_N^2 & I_N^3 \end{pmatrix}$$

During the prediction stage, we divide each image frame into $n \times m$ blocks and estimate the migration velocity for each block using the linear regression. We can then plot the velocity distribution over time. The expectation of the migration velocity can also be estimated for each frame by

$$E(v) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij}$$

where v_{ij} is the velocity in each block.

4. EXPERIEMNTS

4.1 Datasets

Our proposed algorithm is tested on three microcopy image sequences. A few frames of the same cell type are used for estimation of parameters **w**.

Dataset A has one sequence of human umbilical vein endothelial cells (HUVEC) that are isolated from normal umbilical vein [Cel09]. The images are cropped to 474 × 364.

Dataset B includes image sequences that show cell migration of primary keratinocytes before and after calcium switch [Yam09]. These images have a dimension cropped to 642 × 449 pixels.

Dataset C contains one image sequence taken overnight by using a spinning disc confocal microscope [Mar09]. It shows motility of cancer cells. This image sequence has 111 frames with 472 × 360 pixels/frame.

4.2 Estimation of Parameters

The parameters **w** for the linear regression are learned from the training data. For each dataset, the parameters are learned using a set of samples marked by our interactive graphic interface. The image block size is set to 64 × 64, which is around 2 times larger than a typical cell length The training sets include 20~40 samples from each dataset. Figure 4-7 show the scatterplot matrix of the 34 training samples from dataset A, and 25 training samples from dataset C. It is easy to see the strong correlation among mutual information, area ratio, and migration. The training samples from the dataset B have the distribution similar to the figure 4 and 5 from the dataset A.

The area ratio distribution in figure 7 is more dispersed than the distribution shown in figure 5. This is mainly because that the cell intensity is very close to the background intensity as shown figure 1(b). The segmentation based on intensity tends to fail. In this case, the regression with only mutual information can provide a more stable result. Table 1 gives a summary of parameters for dataset A and C.

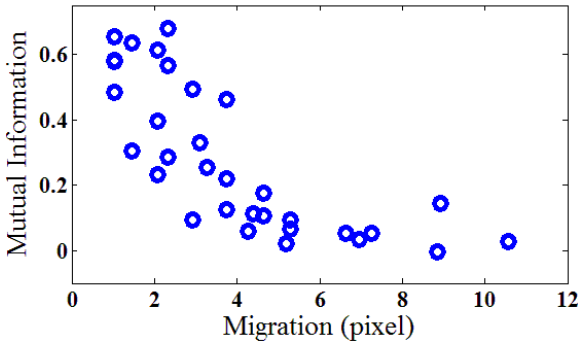


Figure 4. Mutual information *I* versus migration using 34 training samples from dataset A.

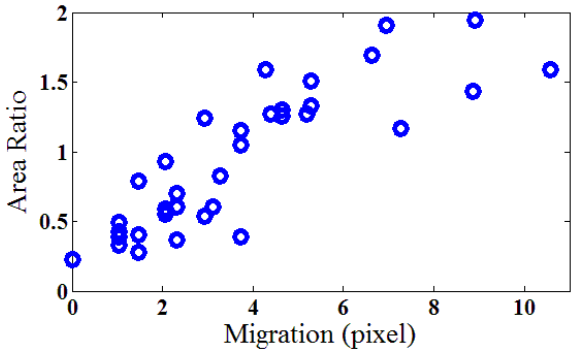


Figure 5. Area ratio *D* versus migration using 34 training samples from dataset A.

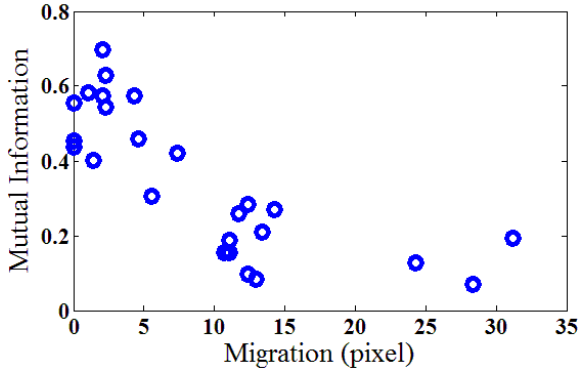


Figure 6. Mutual information *I* versus migration using 25 training samples from dataset C.

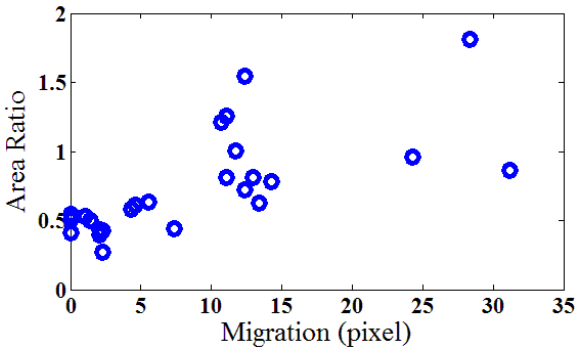


Figure 7. Area ratio D versus migration using 25 training samples from dataset C.

Table 1. Summary of parameters for dataset A and database C.

DB	w_0	w_1	w_2	w_3	w_4	w_5	Block Size
A	0.202	0.246	0.039	0.349	0.016	0.031	64×64
C	0.114	0.041	0.019	0.010	N/A	N/A	64×64

4.3 Estimation of Migration Velocity

After the training stages, we compute the migration velocity over the whole range of each dataset.

Figure 8 shows the result from the dataset A. We computed migration velocity using our algorithm for every 50 frames over 4271 image frames captured over 70 hours and 37 minutes. This velocity distribution with the bell shape is same as our expectation. At the beginning of the captured image sequences, the number of cells is relatively small and the cell growth rate is high. After the growth rate reaches its peak, the cells are very crowded and touch each other in the limited space. As a result, the growth rate decreases. Figure 9 shows two image frames at the beginning and at the end of the dataset A.

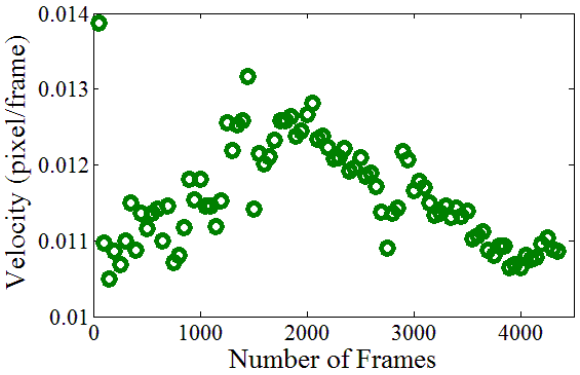
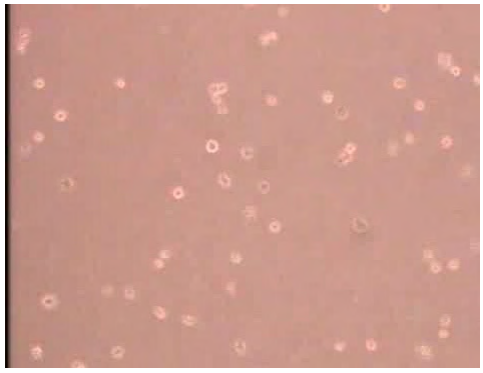
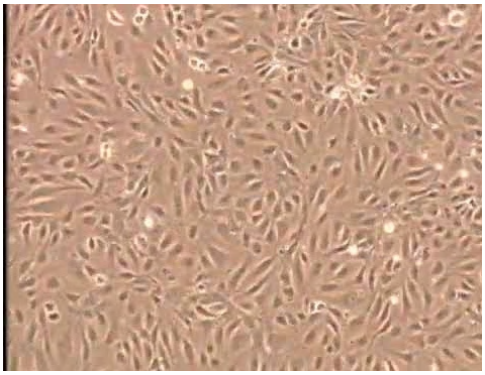


Figure 8. Velocity distribution for the dataset A.



(a) 92th Frame at 01m00s



(b) 4,250th Frame at 68h46m30s

Figure 9. Two image frames from the dataset A that could be used to further verify the cell growth rate.

For the dataset B, as the concentration of calcium was increased from low to high, cell migration velocity is prevented by the maturation of cell-cell adhesion after the calcium switch. Therefore, the velocity decreases shown in figure 10 is also same as our expectation. Here we take every 3 frames to compute velocity over 73 frames that are captured using 6 hours.

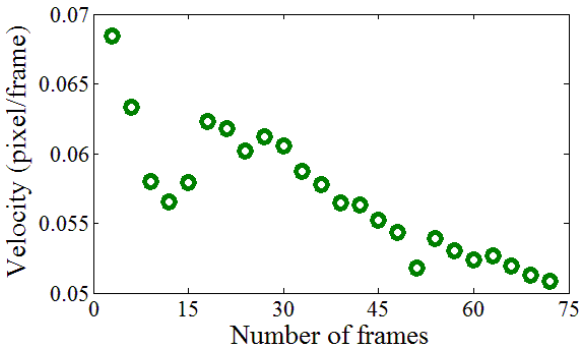


Figure 10. Velocity distribution for the dataset B.

Figure 11 shows the velocity distribution for the dataset C over 111 frames using more than 18 hours.

We sample every 10 frame. As the population of cancer cells gradually reaches its peak in the limited space, the velocity also decreases gradually. We further plot the mutual information distribution for every frame in order to verify our results. Figure 12 shows the distributions and the corresponding image frames. It is clear to see the distributions of mutual information are very similar to the distribution of the migration velocity.

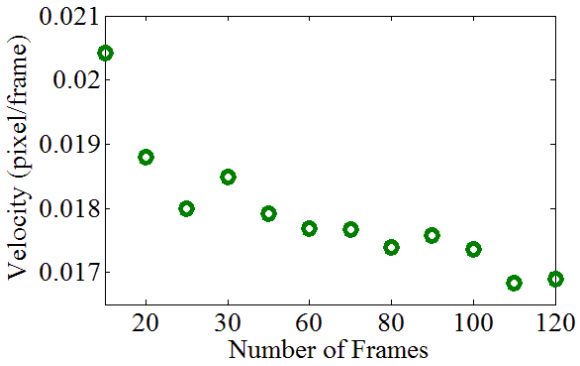
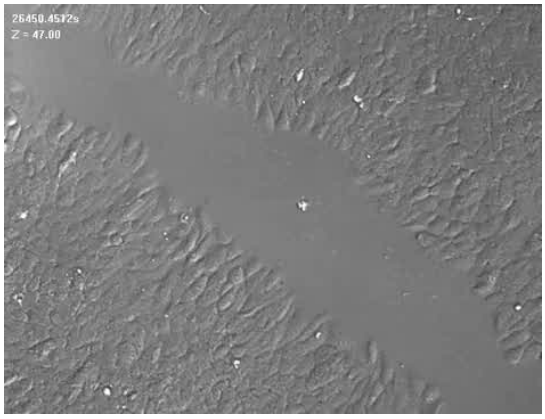
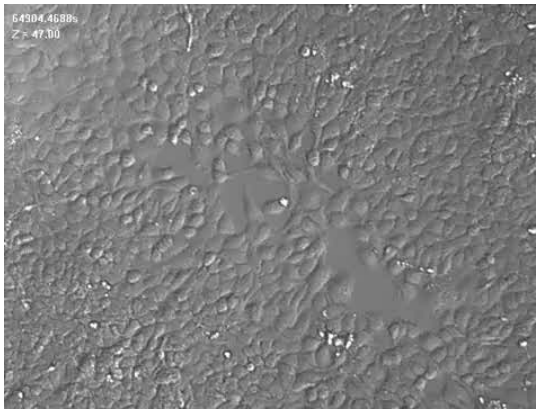


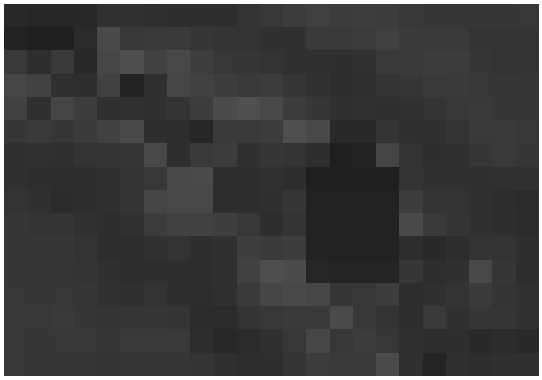
Figure 11. Velocity distribution for the dataset C.



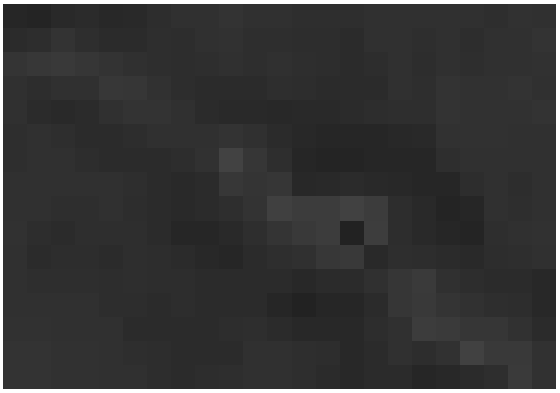
(a)



(b)



(c)



(d)

Figure 12. Images and distributions of mutual information. (a) 45th frame of dataset C. (b) 109th frame of dataset C. (c) distribution of mutual information corresponding to (a). (d) distribution of mutual information corresponding to (b). (c) and (d) are enhanced for the purpose of visualization. The brighter a region, the larger the mutual information is.

5. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel algorithm to estimate cell migration velocity. As individual cell segmentation and tacking are avoided, this algorithm is efficient and robust. Our experiments show that this algorithm is also accurate and can be used to measure cell motility over a long time. In the future, we would like to extend our work to estimation of cell division, merging, and growth based on our regression framework.

6. ACKNOWLEDGMENTS

The authors acknowledge support of the National Science Foundation HRD 0833184.

7. REFERENCES

[Cel09] Cell Applications, Inc., Human Endothelial Cells, http://www.cellapplications.com/endothelial-cells.

[DSA+08] J. G. G. Dobbe, G. J. Streekstra, B. Atasever, R. van Zijderveld, and C. Ince, Measurement of functional microcirculatory geometry and velocity distributions using automated image analysis. Journal of Medical and Biological Engineering and Computing, Volume 46, Number 7, Pages 659-670, 2008.

[HZ04] Hartley, R. and Zisserman, A., Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, ISBN: 0521540518, 2004

[LGM04] Lamberti, F., Gamba, A., and Montrucchio, B, Computer-assisted analysis of in-vitro vasculogenesis and angiogenesis processes. Journal of WSCG, Volume 12, Number 1-3, Pages 237-244, 2004.

[LMC+08] Kang Li, Eric D. Miller, Mei Chen, Takeo Kanade, Lee E. Weiss, and Phil G. Campbell, Cell population tracking and lineage construction with spatiotemporal context. Journal of Medical Image Analysis, Volume 12, Issue 5, Pages 546-566, 2008.

[Mar09] Charles Marcus, Cancer cell migration and movement, http://marcuslab.harvard.edu/index.shtml, Department of Physics, Harvard University.

[MDI+09] E. Meijering, O. Dzyubachyk, I. Smal, and W.A. van Cappellen, Tracking in Cell and Developmental Biology. Seminars in Cell and Developmental Biology, Volume 20, Issue 8, Pages 894-902, 2009.

[Otu79] Otsu, N., A threshold selection method from gray level histograms. IEEE Transactions on Systems, Man, and Cybernetics - Part B, Volume 9, pages 62–66, 1979.

[PMV03] Pluim, J.P.W., Maintz, J.B.A., and Viergever, M.A., Mutual-information-based registration of medical images: a survey, IEEE Transactions on Medical Imaging, Volume 22, Issue 8, Pages 986-1004, 2003.

[WZH+09] Chih-Chieh Wu, Geoffrey Zhang, Tzung-Chi Huang, and Kang-Ping Lin, Red blood cell velocity measurements of complete capillary in finger nail-fold using optical flow estimation. Journal of Microvascular Research, Volume 78, Issue 3, Pages 319-324, 2009.

[Yam09] Soichiro Yamada, Keratinocyte migration, http://yamadalab.ucdavis.edu/, Biomedical Engineering, University of California at Davis.

[YPW+04] Justin C Yarrow, Zachary E Perlman, Nicholas J Westwood, and Timothy J Mitchison, A high-throughput cell migration assay using scratch wound healing, a comparison of image-based readout methods, BMC Biotechnology, Volume 4, Number 1, 21, 2004

Caustic Object Construction Based on Multiple Caustic Patterns

Budianto Tandianus
Nanyang Technological University, Singapore
budi0010@ntu.edu.sg

Henry Johan
Nanyang Technological University, Singapore
henryjohan@ntu.edu.sg

Hock Soon Seah
Nanyang Technological University, Singapore
ashsseah@ntu.edu.sg

ABSTRACT

Inverse caustic problem, that is computing the geometry of a reflector and/or refractor based on a given caustic pattern, is currently not widely studied. In this paper, we propose a technique to solve the inverse caustic problem in which we compute the geometry of a semi-transparent homogeneous refractive object (caustic object) given a directional light source and a set of caustic patterns (each pattern is considered to be formed at a specified distance from the caustic object). We validate the results by using mental ray (software rendering). The novelty of our research is that we consider a set of caustic patterns whereas existing techniques only consider one caustic pattern. We employ a stochastic approach to simulate the refracted light beam paths that can approximately reconstruct the input caustic patterns. Working backward, from the computed refracted light beam paths we compute the geometry of the caustic object that can produce such light beam paths. Due to having multiple caustic patterns as the inputs, it is a challenge to reconstruct the input caustic patterns because of the differences in their shapes and intensities. We solve this problem by using a two-step optimization algorithm in which we adjust the position and size of the caustic regions in the first step and we adjust the caustic shapes in the second step. Our technique is able to construct a caustic object for a various types of input caustic patterns.

Keywords: caustics, photon, reconstruction, inverse problem, stochastics

1. INTRODUCTION

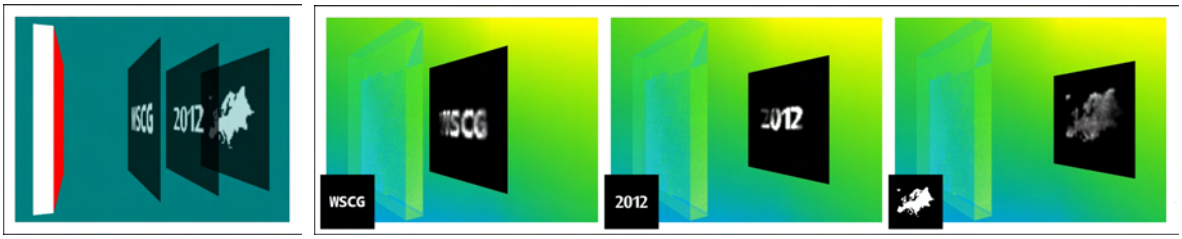
Recently, there is a growing interest in inverse problem research in Computer Graphics due to the possibility of controlling the creation of visual effects. By using the inverse techniques, the design process becomes easier as the artists can just specify the intended effects directly instead of performing the iterative trial-and-error process. However, inverse problem is generally difficult as in most cases there is no unique bijective relationship between the output and the input (i.e., given an output, there are many input possibilities that can generate such output).

In the inverse caustic problem, given an input caustic pattern (shape, intensity, and location from the caustic object) and a light source, we have to compute the geometry of the caustic object that can produce a caustic pattern similar to the input caustic pattern. Inverse caustic problem is hard to solve because the input caustic pattern only contains the irradiance magnitude and it does not have incident light direction information (and also the reflected and/or refracted light paths). Up to now, the inverse caustic problem is not widely studied and the existing work only consider a single input caustic pattern.

In this paper, we propose a new inverse caustic problem, that is computing the caustic object given multiple input caustic patterns formed on a caustic receiver (diffuse and non-transparent surface) at various distances from the caustic object. We show an example in Figure 1.

Our basic idea for solving this problem is as follows. We subdivide one side of the caustic object and also the caustic patterns into regular cells. The light beam refracted by each caustic object cell will pass through one caustic cell of each caustic pattern. We try to compute the orientation of each caustic object cell such that the combination of the refracted light beams of all caustic object cells can approximately reconstruct the input caustic patterns.

We use a stochastic approach in our technique and we represent each input caustic pattern as a 2D probability mass function (pmf) by considering the brightness of a caustic cell as the probability of a light beam might pass through it (i.e., the brighter the caustic cell is, the higher probability or the more likely a light beam is considered to pass through it). Hence, for each cell of the caustic object, we use the pmfs of the caustic patterns to determine to which direction the caustic object cell refracts a light beam. From the determined refracted light beam



(a) Scene setup. (b) Computed caustic object and approximately reconstructed caustic patterns.

Figure 1: (a) Scene setup. We compute a caustic object (the leftmost box), specifically the surface geometry of the side (shown in red color) facing the caustic patterns, given three caustic patterns (WSCG, 2012, and Europe) to be formed on a caustic receiver at three distances from the caustic object, with a directional light source orthogonal to the caustic object illuminates from the left. (b) mental ray renderings of caustics produced by our computed caustic object (final output). Input caustic patterns are shown in the insets at each image. The computational time is 9.0 hours.

direction, we can compute the orientation of the caustic object cell.

Due to differences of the input caustic patterns (in terms of shapes and intensities), it is hard to compute the caustic object that can satisfy all the input caustic patterns. Thus, we relax the input requirements by slightly adjusting the sizes, positions, and shapes of the non-zero intensity regions of the input caustic patterns. Moreover, we also allow a small amount of light beam which has passed through several caustic patterns to miss or overshoot the rest of the caustic patterns. We compute these adjustments by using optimization techniques. We validate our results by performing rendering simulation using mental ray [men12a], a robust industry standard rendering engine.

2. RELATED WORK

Unknown Input Given only the output, the input that can produce such output is computed. This is a hard problem since the *a priori* knowledge of the input is not available. One example is the work presented by Bottino et al. [Bot01a] and Mitra et al. [Mit09a]. They compute a 3D geometry that can satisfy the inputs which consist of a set of silhouettes or shadow patterns.

Inverse Caustics One of the earliest work in inverse caustic is presented by Patow and Pueyo [Pat04a]. They compute the reflector shape in an optical set (consists of a reflector, light source, and diffusor) given the radiance distribution as an input. They represent the reflector as grids and they iteratively adjust the grid vertices such as positions and number of vertices based on the similarity with the intended radiance distributions. The whole process took many days even though they can obtain radiance distribution similar to the input. They improve the work by allowing the user to set the range of the solution space [Pat07a] (the lower bound and the upper bound of the reflector shape). Hence, a user has more control in determining the reflector shape. They later increase the performance by using GPU [Mas09a] and they can reduce the processing time into magnitude of hours.

In parallel with the aforementioned work, Anson et al. [Ans08a] represent the reflector as a NURBS surface and Finckh et al. represent the reflector as a B-Spline surface [Fin10a]. As a result, during the optimization they optimize the control points instead of grid vertices which in the end can produce smooth reflectors in a relatively fast speed (due to the small number of parameters to be optimized). However, as Papas et al. also mention [Pap11a], the parameterized technique has a difficulty with highly complex caustic images, thus Finckh et. al [Fin10a] cannot reproduce all frequencies of the caustic pattern and Anson et. al [Ans08a] assume the shape of the caustic pattern to be circular.

Weyrich et al. generate a microgeometry reflector given a single reflected caustic pattern input [Wey09a]. The caustic object is subdivided into uniform cells (facets), and they compute the optimized orientation of each cell that can produce a caustic pattern similar to the input pattern. Papas et al. [Pap11a] improve the work of Weyrich et al. [Wey09a] by generating a refractor caustic object on a larger scale. Moreover, they are able to prevent noise on the reconstructed caustic pattern by computing the surface of each facet based on the Gaussian distribution. Similar to Weyrich et al. [Wey09a], they employ several optimization costs in order to generate the caustic objects. In the most recent development, Yue et al. [Yue12a] emphasize on modularity by reconstructing an input caustic pattern from a caustic object which consists of many smaller pieces of caustic object cells. Their caustic object cells are divided into ten types with each type refracts light to a predefined direction.

Comparison In all these work, the input is only a single caustic pattern. On the other hand, in this paper we propose a new challenge in which we compute the geometry of a caustic object based on a set of input caustic patterns.

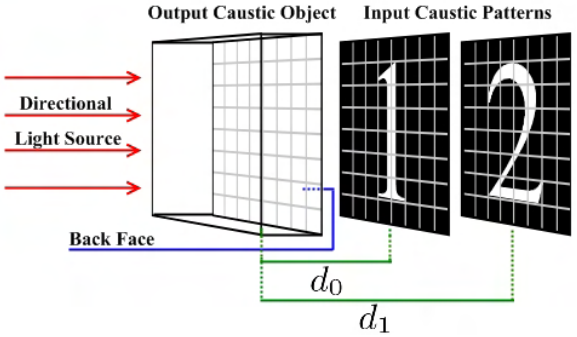


Figure 2: Scene setup. Our algorithm computes the normal/orientation of each caustic object cell. Caustic pattern '1' is formed when the caustic receiver is at distance d_0 from the caustic object and similarly caustic pattern '2' at d_1.

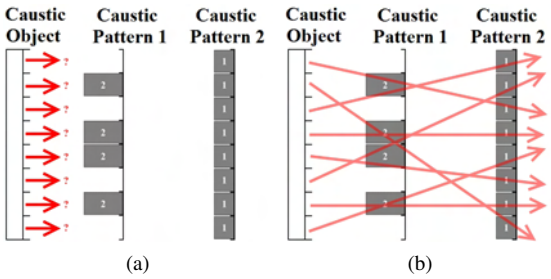


Figure 3: Problem formulation. (a) Given two caustic patterns at two different locations (with the intensity of each caustic cell is denoted by the size of the cell), compute light refraction direction (red arrow) of each caustic object cell such that the refracted light collectively can generate caustic patterns similar to the input caustic patterns. (b) Light refraction combination that can satisfy the input caustic patterns.

3. BASIC IDEA OF OUR METHOD

Scene Setup The scene setup is shown in Figure 2. The scene consists of three components, a caustic object (of a box shape), a caustic receiver (a planar surface where the caustic patterns are formed), and a directional light source (whose direction is orthogonal to the caustic object). Both the caustic receiver and the caustic object are positioned coplanar, with the caustic receiver is on one side of the caustic object (assumed to be the **back face** of the caustic object, facing (0, 0, -1) direction) and incoming light direction is on the other face of the caustic object (assumed to be the front face of the caustic object, facing (0, 0, 1) direction). We assume the caustic receiver and the caustic object to have the same spatial dimension (i.e. same width and height) and orientation. Therefore, the extent of the region of interest of each caustic pattern is bounded by the shape of the caustic receiver

The caustic patterns and the back face of the caustic object are subdivided into a regular grid of cells. Each cell of a caustic pattern stores the total caustic intensity on that particular cell. We call cells of caustic patterns which have non-zero caustic intensity as **caustic cells** and cells with zero caustic intensity as **empty cells**. The collection of caustic cells of a caustic pattern is collectively called as a **caustic region**. For each cell of the caustic object (**caustic object cell**), we compute its orientation such that it can produce a refracted light beam to a specific direction. In the rest of this paper, we refer to each refracted light beam as light and we represent each light beam in the following Figures 3 and 4 as an arrow.

Problem Formulation We compute the back face of a caustic object C given a set of p grayscale caustic patterns such that each caustic pattern j will be formed on the caustic receiver when the receiver is located at the user-input distance d_j from the caustic object. Specifically, we compute the orientation of each caustic object cell at the back face of the caustic object such that the cell refracts the incoming light into a direction that passes through parts of the caustic regions. Collectively, the light refracted from all caustic object cells is expected to pass through all the input caustic cells thus reconstructing the input caustic patterns. As mentioned in Section 1, the input caustic patterns only provide the estimate of the amount of refracted light arriving at caustic receiver cells, not the light directions. Hence, the main challenge is to compute refracted light paths that can approximately reconstruct all the given caustic patterns. This problem is illustrated in Figure 3. The orientation of each cell can then be determined based on the path of its refracted light.

Solution As explained above, the task is to compute refracted light direction combinations such that they can approximately reconstruct the input caustic patterns. In this case, more light is expected to pass through brighter caustic cells compared to darker caustic cells. Hence, to solve this, we simulate the direction of the refracted light of each caustic object cell by using a stochastic approach. The idea is to use the caustic intensity in each caustic cell as the probability that we will refract a light to that caustic cell (i.e. the brighter the input caustic pattern is, the more likely it is chosen as a refracted light target).

We represent the set of caustic patterns as a set of normalized 2D probability mass functions $\mathbf{P} = \{f_{P_0}, f_{P_1}, f_{P_2}, \ldots f_{P_{p-1}}\}$ (each in $\mathbf{P}$ is the pmf of the user-input caustic pattern on the caustic receiver when the receiver is located at the user-input distance from the caustic object). The pmf of each caustic pattern is defined by using the grayscale value (intensity) of the caustic pattern in which the probability at each caustic cell is the grayscale value of that particular cell in the caustic pattern. Each pmf f_{P_j} is normalized by

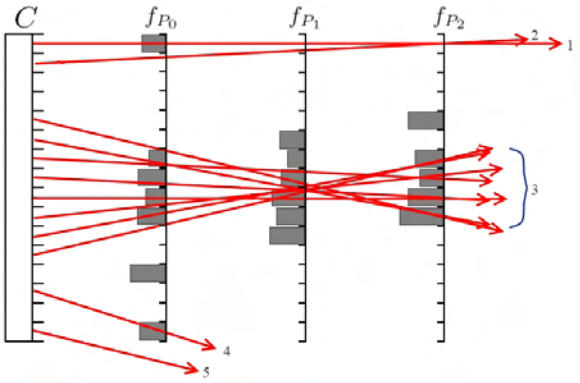


Figure 4: Each numbered arrow denotes the refracted light from the cells of the caustic object C and the gray blocks denote the probability of caustic cells. Light #1 and #2 have joint pmf of zero since their paths pass through at least one empty cell. Each light in #3 has the probability greater than zero since the light pass through non-zero cells. Light #4 is allowed even though it misses some of the caustic patterns. We will explain this further in Section 4.2. Light #5 is not valid because it does not intersect any caustic patterns

dividing the probability of each of its cell with the total probability of all cells of f_{P_j}.

We assign a random variable X_i for each i-th cell of the caustic object. For each X_i, the probability value of each possible refracted light direction $\mathbf{x}$ is computed by multiplying the probability of each caustic cell passed by the light refracted to direction $\mathbf{x}$ (see Figure 4), as shown in Equation 1.

$$f_{X_i}(\mathbf{x}) = \prod_{j=0}^{p-1} f_{P_j}(g_j^i(\mathbf{x})), \tag{1}$$

with $g_j^i(\mathbf{x})$ is a mapping function. The mapping function is basically a ray casting function in which the light is shot from the i-th caustic object cell to the caustic pattern j with the direction of $\mathbf{x}$ and return the intersected cell (of caustic pattern j). Then, for each caustic object cell or each X_i we assign a refracted light direction by using the Acceptance-Rejection method [vN51a] with the distribution based on the sampled joint probability mass function of all caustic patterns (Equation 1).

Once we obtain the refracted light direction for each caustic object cell, we compute its normal or orientation based on the user-input index of refraction of the caustic object, incoming light direction (orthogonal to the caustic object), and the obtained refracted light direction by solving the Snell's Equation (see Appendix A). Afterward, we perform rendering simulation using the mental ray to asses the approximate reconstructed caustic patterns (as shown in Figure 6a).

Note that our technique can also be applied to point light sources and directional light sources non-orthogonal to the caustic object. In the rest of the paper, the terms caustic patterns and pmfs are interchangeable as we use the term pmfs when we emphasize on the mathematical representation of the caustic patterns.

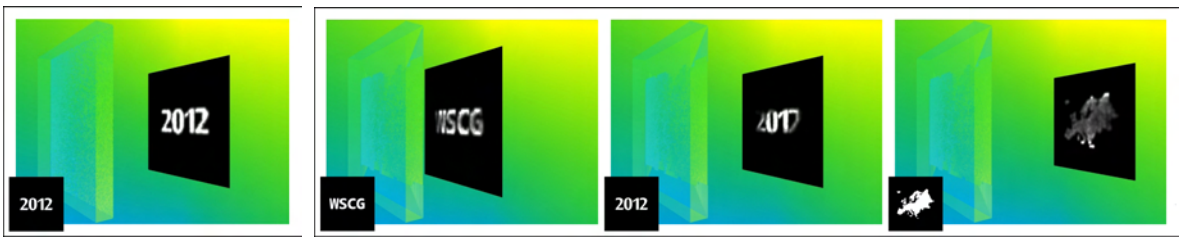
4. IMPROVING THE RECONSTRUCTED CAUSTICS

The solution in Section 3 may not be able to reconstruct the caustic patterns very well if the input consists of multiple patterns. If we only have a single caustic pattern (shown in Figure 5a), then we can reconstruct it very well. However, if we add two additional caustic patterns, then some parts of the input caustic patterns are missing (Figure 5b).

Reconstruction problem As explained in Section 3 (and shown in Figure 4), some refraction directions have zero joint pmf when they pass through at least one empty caustic cell. As a result, if all possible refraction directions from every caustic object cell pass through a caustic cell of a caustic pattern but they also pass through the empty cells of other caustic patterns, then the aforementioned caustic cell cannot be reconstructed (we call such cell as a **missing caustic cell**). As seen in Figure 4, the top and bottom caustic cells in f_{P_1} are missing caustic cells since the refracted light that pass through these caustic cells also pass through empty cells in the other caustic patterns.

Proposed solution Based on the given input caustic patterns and their configurations (positions and sizes), it might not be possible to compute the caustic object that can well reconstruct the original input caustic patterns. Thus, we propose a method to relax the input requirement by allowing slight changes to the positions, sizes, and shapes of the caustic regions. Our proposed method consists of two steps. In the first step, we optimize the size and position of the caustic regions by slightly adjusting the size and position given by the user (Section 4.1). In the second step, the boundaries of each caustic region are adaptively extended such that they enable the reconstruction of the missing caustic cells on the other caustic patterns (Section 4.2). We also compute the amount of light that is allowed to overshoot or to miss some caustic patterns.

In both optimization steps, we use Simulated Annealing [Kir83a]. The main reason we use Simulated Annealing is because the problem cannot be solved analytically. However, there are also some possible optimization techniques such as Particle Swarm, Ant Colony, and Genetic Algorithm. However, those techniques require keeping the record of multiple possible solutions at once, hence it is not efficient for our case (as seen in Equation 3, the cost computation requires reconstruction of the caustic patterns multiple times using the adjusted input caustic patterns). Moreover, in some of the related work [Wey09a, Fin10a, Pap11a], Simulated Annealing is also used. After applying our proposed



(a) (b)

Figure 5: (a) Only a single caustic pattern and it can be reconstructed very well. (b) Two additional caustic patterns cause some parts of the input caustic patterns to be missing.

solution, the reconstructed caustic patterns from the test case in Figure 5b are improved as seen in Figure 1b.

Cost computation In every iteration of both optimizations, we use the root mean square to compute the cost or degree of possibility that the adjusted input caustic pattern can be approximately reconstructed (in order to guide the simulated annealing). The root mean square is computed as the difference between the normalized reconstructed caustic patterns $\mathbf{Z} = \{f_{Z_0}, f_{Z_1}, \dots f_{Z_{p-1}}\}$ and the normalized adjusted input caustic patterns $\mathbf{D} = \{f_{D_0}, f_{D_1}, f_{D_2}, \dots f_{D_{p-1}}\}$. Only in the cost computation here, the normalization of caustic patterns in $\mathbf{Z}$ and $\mathbf{D}$ are computed by dividing the value of each caustic cell $\mathbf{t}$ with the maximum caustic cell value of the caustic pattern $\mathbf{t}$ belongs to. We compute the cost in this way such that the maximum cost (which is in the worst case scenario, for example the input caustic patterns are not reconstructed at all) is 1.0. The cost computation is shown in Equation 3.

$$Cost = \frac{1}{p}\sum_{j=0}^{p-1}\sqrt{\frac{1}{n(\mathbf{W}(j))}\sum_{i=0}^{\beta}(f_{D_j}(\mathbf{t}_i) - f_{Z_j}(\mathbf{t}_i))^2}, \quad (2)$$

with

$$\mathbf{W}(j) = \{\mathbf{t} | f_{D_j}(\mathbf{t}) + f_{Z_j}(\mathbf{t}) > 0\}, \quad (3)$$

and $\mathbf{t}$ is the caustic cell, p is the number of caustic patterns, $n(\mathbf{W}(j))$ is the number of elements of a set of caustic cells contributing to the cost computation, and β is the total number of cells of each caustic pattern (in our experiments, $\beta = 64 \times 64 = 4096$). For more accurate computation of $\mathbf{Z}$, we approximately reconstruct the caustic patterns 32 times and accumulate their caustic cell values, and finally we divide the value of each caustic cell by 32 in order to get $\mathbf{Z}$.

We reconstruct the caustic patterns by computing the refracted direction as explained in Section 3 and then for each caustic cell we accumulate the amount of refracted light that intersects it. We use the modified pmfs (which are adjusted in each optimization step) to compute the joint pmfs (Equation 1).

4.1. Adjusting the Size and Position

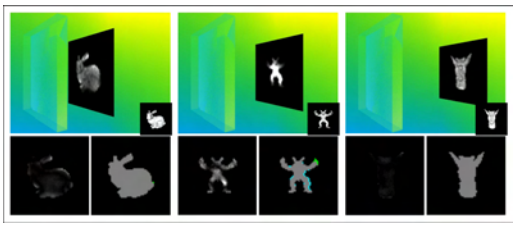
In this first optimization step, we relax the input caustic pattern configurations by iteratively adjusting the size and position of the input caustic region. Adjusting the position is basically translating the caustic regions in 3D space (translation in x, y, z). This means we also adjust the input distance (translation in z) between the caustic object and the caustic pattern (caustic receiver).

In every iteration, we adjust the size and positions of the input caustic regions and compute the cost by using Equation 2 in order to guide the optimization iterations. The adjusted input caustic patterns are used as the input to the next optimization step (Section 4.2) and they are also the target caustic patterns for every iteration of the second step.

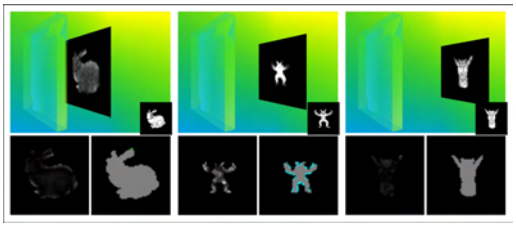
4.2. Extending Caustic Regions and Overshooting Refracted Light

After performing the first optimization step, there might be some missing caustic cells left. Missing caustic cells are the caustic cells that cannot be reconstructed. To reconstruct some of these missing caustic cells, we slightly extend the shape of all input caustic regions. For example, in Figure 4, the middle-top and middle-bottom caustic cells of f_{P_1} cannot be reconstructed since all possible refracted light that passes through these cells in f_{P_1} have to pass through empty cells in either f_{P_0} or f_{P_2}. Hence, to solve this, we can extend the middle caustic regions in f_{P_0} and f_{P_2} up and down by one cell.

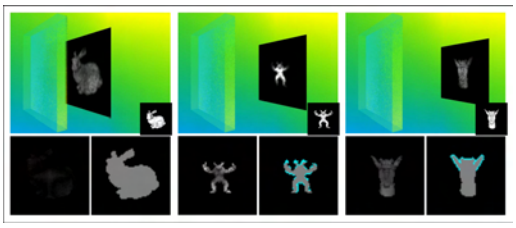
Some caustic cells especially around caustic patterns borders are hard to reconstruct as most light passing through these cells can miss other caustic patterns in behind. Thus, we relax this requirement by enabling some of the refracted light that passes through caustic cells of one or several caustic patterns to miss the rest of the caustic patterns (or region of interests of the rest of the caustic patterns). This is beneficial especially for the caustic cells on the border of the caustic patterns. For example, in Figure 4, if we enable light #4 to pass the bottommost caustic cell of f_{P_0} and miss the rest of caustic patterns (we call this **overshoot**), then



(a) Results without optimization. Cost : 4.45×10^{-1}



(b) Results after 1st optimization (Section 4.1). Cost : 4.08×10^{-1}

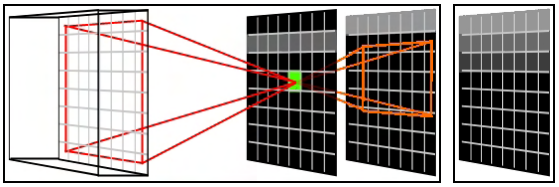


(c) Results after 1st and 2nd optimization (Section 4.2). Cost : 2.64×10^{-1}

Figure 6: Mental ray rendering results of the optimization steps. Input caustic patterns are shown at the bottom right of each screenshot in (a). We also show the missing caustic cell maps at the below right of each image (green cells show the missing caustic cells, gray cells show the caustic cells that can be reconstructed, and cyan cells show the extended caustic cells). For the visualization of the differences between the target and the reconstructed caustic patterns, we also show the caustic irradiance difference maps (assuming the total irradiance of each target caustic pattern is 1.0 and the total light emitted to the scene is 1.0, i.e. each caustic object cell refracts the light with the amount of 1.0 divided by the number of caustic object cells) at the below left of each image (from the darkest pixels with the least errors to the brightest pixels with the most errors). For the sake of visual clarity, we scale up the difference values by 5000. The computational time is 5.7 hours.

the bottommost caustic cell of f_{P_0} can be reconstructed. However, we still do not allow the refracted light of one caustic object cell to miss all of the caustic patterns (as in light #5).

Fully extending the caustic regions can deform the original caustics too much, and likewise if we allow too much light to overshoot the caustic patterns then the approximate reconstructed caustic patterns will have very low intensity. Hence, in this step, we apply an optimization to determine the appropriate caustic regions extensions amount $\mathbf{k} = \{k_0, k_1, \ldots k_{p-1}\}$ and light overshoot amount $\mathbf{o} = \{o_0, o_1, \ldots, o_{p-1}\}$ with $\mathbf{k}$ and $\mathbf{o} \in [0,1]$ (i.e., a k and an o value for each caustic pattern).



(a) (b)

Figure 7: (a) A simple example of missing caustic cell projection (is explained in Section 4.2). Gray cells are the caustic cells and the green cell is the missing caustic cell. (b) We extend the second caustic pattern (two cells away) with gradually decreasing intensity.

Extending Caustic Regions To enable the missing caustic cells of a caustic pattern j to be reconstructed, we have to firstly compute at most how many s_b unit cells away the caustic region boundaries of the other caustic patterns b ($0 \leq b \leq p-1, b \neq j$) have to be extended. Afterward, for every caustic pattern b, we extend its caustic region with the amount of $s_b \cdot k_b$. In order to enable smooth extension, we extend the caustic regions with linearly decreasing intensity (or probability value).

To do this, for every missing caustic cell of the caustic pattern j, we project it from every caustic object cell to the empty cells of other caustic patterns b ($0 \leq b \leq p-1, b \neq j$). We perform this projection for the missing caustic cells of all caustic patterns. This projection example is shown in Figure 7a in which we project the missing caustic cell (shown in green color). Afterward, for each caustic pattern b, we obtain the maximum distance (s_b) between its caustic region boundaries and its empty cells that receive the projections of the missing caustic cells (of other caustic patterns). In Figure 7a example, it is five cells ($s_b = 5$) away for the second caustic pattern and in Figure 7b the caustic region boundary is extended two cells away (if $k_b = 0.4$).

Some of the missing caustic cell projections might miss the other caustic patterns b. For example, if the missing caustic cell in Figure 7a is one or two cells to the right, then some of the projections will overshoot or will not hit the second caustic pattern. We use this information to control the possibility of the refracted light to overshoot each caustic pattern.

Overshooting Refracted Light To improve the results, we also enable the refracted light to overshoot some of the caustic patterns. Thus, during the missing caustic cells projections, we also compute the ratio (e_b) between the amount of these projections that do not hit caustic pattern b and the total amount of these projec-

tions toward caustic pattern b (i.e. sum of the missing caustic cell projections that hit and do not hit caustic pattern b). e_b is essential since it provides the information on the amount of probability that the refracted light miss plane b. Hence, the probability h_b that we will refract the light to miss the caustic pattern b is shown in Equation 4.

$$h_b = e_b \cdot o_b \cdot f_{P_b}(\mathbf{t}_{max}), \tag{4}$$

with o_b is the coefficient to control the probability of overshooting b-th caustic pattern and $\mathbf{t}_{max}$ is a cell of f_{P_b} with the highest probability value.

We show the optimization progression in Figure 6.

5. GEOMETRY CONSTRUCTION

As explained in Section 3 we use the joint pmf (Equation 1) of the caustic patterns to compute the refraction direction of each caustic object cell. From the refraction direction, we can obtain the normal of the particular caustic object cell. However, if we perform the optimizations in Section 4, then we use the modified pmfs (output from both optimization steps) to compute the joint pmf (and ultimately the normal of each caustic object cell).

Based on the computed orientation of each caustic object cell, we can obtain the caustic object geometry by computing the x,y,z coordinates of the four corners of each caustic object cell. The x,y coordinates of each caustic object cell corner can be easily found as the caustic object is uniformly subdivided. To compute the z coordinates of the caustic object cell corners, we firstly assume that the z coordinate of all caustic object cell middle points to be the same ($z = 0.0$). Next, we use the dot product operation, i.e. dot product between the normal of the caustic object cell and the vector from the caustic object cell middle point (we know its x,y coordinates from the uniform subdivision and also its z coordinate which is 0.0) to each caustic object cell corner (we know its x,y coordinates) must be equal to zero. In this case, the only unknown value is z of the caustic object cell corner.

Since the edges of the neighbouring caustic object cells do not have the same slope or the same z coordinate on both endpoints, there are vertical open spaces between caustic object cells (shaded with lighter gray in Figure 8), we generate additional polygons to close those gaps.

6. RESULTS

We present some results computed using our technique in Figures 1, 6, 10. In all of the test cases, the index of refraction of the caustic objects are 1.5, the resolution of the caustic objects are 128×128 and the resolution of the caustic receiver is 64×64. Resolution refers to

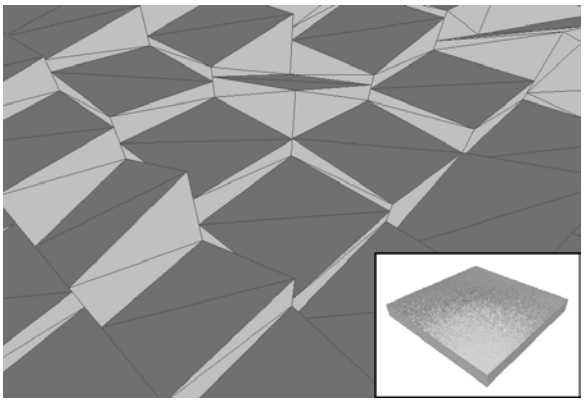
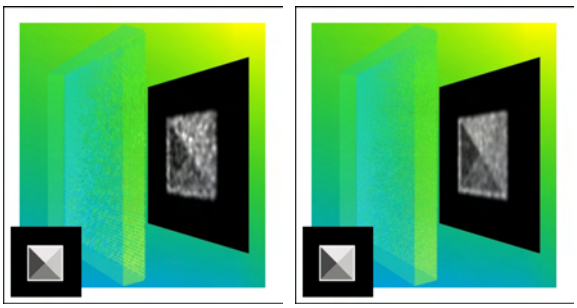


Figure 8: Caustic object geometry (inset) of Figure 6c with a zoom-in view. In the zoom-in view, each caustic object cell consists of two co-planar triangles shaded with darker gray. We also generate additional vertical polygons (shaded with lighter gray) to close the gaps between caustic object cells.

the number of cells. If we assume the spatial size to be 1.0×1.0, then the size of each caustic object cell is $1.0/128 \times 1.0/128$ and the size of each caustic cell is $1.0/64 \times 1.0/64$.

We use higher resolution for caustic object cells since we want to have more variations on the refracted light paths so that we can better reconstruct the contrast (or intensity variations) of the given caustic patterns. We show a difference example with a single caustic pattern case in Figure 9, a simple caustic pattern (resolution 64×64) reconstructed with a resolution 64 x 64 caustic object and a resolution 128 x 128 caustic object.



(a) (b)

Figure 9: A simple test case (one caustic pattern, with resolution 64×64) reconstructed with different caustic object resolutions. (a) Resolution 64 x 64 caustic object. (b) Resolution 128 x 128 caustic object.

We use the same parameters for the simulation annealing for both optimization steps in all experiments, i.e. 10 cycles of 10 iterations, Boltzmann's constant of 1.0, and temperature reduction factor of 0.5.

The experiments were performed on two comparable PCs. The specification of the first PC is Intel i7 920 2.67 GHz (CPU) with NVIDIA GeForce GTX 285

(GPU) and the specification of the second PC is Intel i7 880 3.07 GHz (CPU) with NVIDIA GeForce GT 330 (GPU). In the implementations, we calculate the joint pmf by rendering each caustic pattern and then we multiply them by using alpha blending (hence the use of GPU). For the rest of the computations such as Simulated Annealing and Acceptance-Rejection method, we perform them on CPU.

From the results, we can observe that the caustic objects generated using our technique can approximately reconstruct various types of input caustic patterns, especially for the WSCG (Figure 1b), fruits (Figure 10a), and rotating star (Figure 10c) test cases. The degree of difficulty in reconstructing the caustic patterns mostly depends on the number of caustic patterns, similarity between shapes, and the number of caustic cells in the input caustic patterns. As shown in Equation 4, due to the multiplication in computing the joint pmf, the probability of a particular refraction direction can become zero if the refracted light passes through empty cells. With the increasing number of caustic patterns especially the ones with different shapes, the chances that we have many refraction directions with zero probability also increase. We can see this from the results in Figures 1 and 6 (three input caustic patterns) where we can reconstruct better compared to the results in Figures 10 (four or more caustic patterns).

The shapes and orientations of caustic patterns can also affect the reconstruction difficulty. The test cases with similar caustic patterns, can be approximately reconstructed pretty well since similar refracted light paths are sufficient to reconstruct the caustic patterns. This is evident by comparing Figure 10a and Figure 10b. The caustic patterns in Figure 10a have near round shapes and approximately the same orientations. In contrast, the test case in Figure 10b have pretty different shapes (and orientations). Hence, there are few light that can pass through the endpoints of the bars compared to the middle regions of the bars.

Note the difficult test case shown in Figure 10b, four bar caustic patterns with alternating orientations. As we can see, the hardest parts to reconstruct are the ones near the two endpoints of the bars and on the other hand the parts around the centers are the easiest. This is due to the alternating shapes which cause the light paths to always pass through the center of the caustic regions. As we allow the refracted light to miss some of the caustic patterns, we are able to approximately reconstruct the top and bottom parts of the first caustic pattern. However, the consequence is that the last caustic pattern appears much dimmer.

In many cases, light tends to converge to the middle caustic patterns and as a result the center regions of these caustic patterns become relatively brighter compared to the other caustic patterns (for example, the Armadillo caustic pattern in Figure 6 exhibits this effect). This is due to two reasons. First, the caustic regions are positioned approximately at the center of the caustic patterns. Second, because the size of the caustic regions are mostly smaller than the caustic object. Therefore, some caustic object cells have to refract the light in the diagonal directions. This is illustrated in Figure 4 in which some of the light grouped to #3 have to be refracted in the diagonal directions.

Note that the relative depth of caustic patterns also affects the quality, as it is very difficult to reconstruct if the caustic patterns are located very near to each other. This is because the refracted light paths will intersect the patterns at similar locations and thus it is difficult to reconstruct caustic patterns with different shapes.

Please refer to the submitted **video** to see the progressive changes of the caustic patterns as the caustic receiver is moved.

7. APPLICATIONS

The inverse caustics has several potential applications.

Arts As shown in Figure 10, our technique can generate caustic objects that can produce several interesting caustic effects (similar to the intention in the inverse shadow [Mit09a]). Therefore, we hope that our work can encourage more exploration in caustic arts.

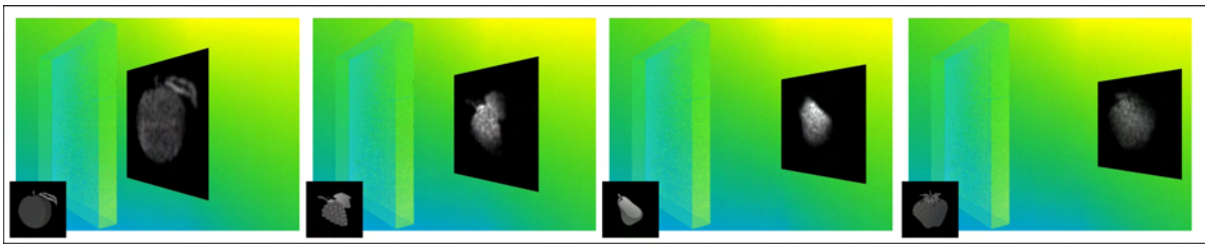
Information Encoding Information (such as serial numbers, passwords) can be encoded as caustic patterns of encrypted 2D images. Only when the requirements (such as light direction and caustic receiver distance) are known, we can recover the original information. We show an example in Figure 11 in which we encrypt **WSCG** and **2012** into two QR barcode patterns.

Validation Tests By using the computed caustic object, we can validate some processes such as rendering process (validating the correctness of caustics rendering algorithm) or manufacturing (validating the quality of produced glasses or light sources).

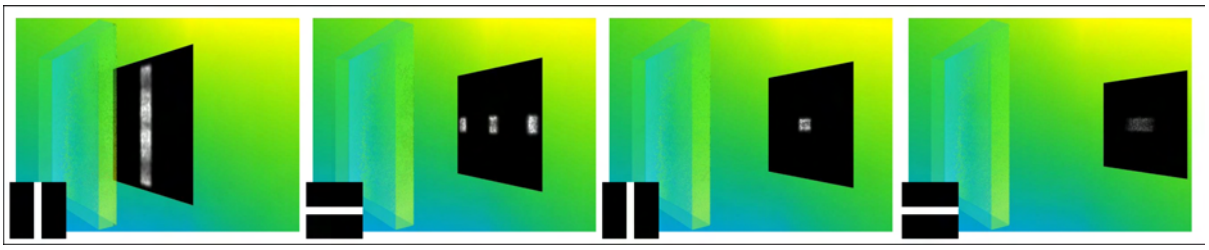
8. CONCLUSIONS AND FUTURE WORK

We have presented an inverse caustic problem and a novel technique which computes a caustic object given a set of caustic patterns with each pattern is positioned at a user-input distance from the caustic object. Our proposed technique is based on a stochastic approach, and it is augmented with two optimization steps that can alleviate the missing caustic problems. We have validated our results by performing physically rendering simulation using mental ray, and the caustic object generated using our technique can approximately reconstruct various types of input caustic patterns.

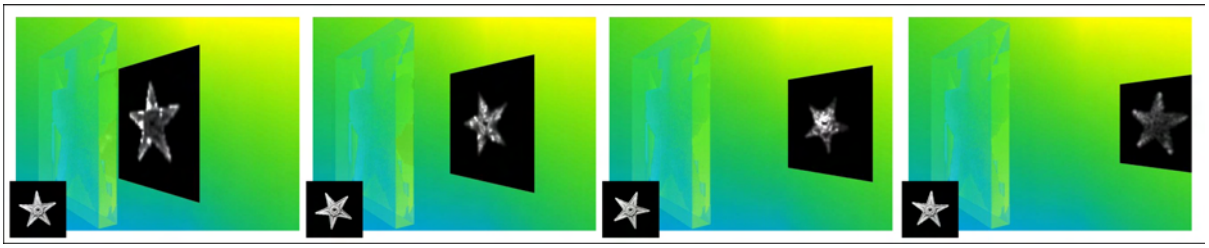
In the future, we would like to improve the quality of the reconstructed caustics in terms of smoothness. It



(a) Fruits (four caustic patterns). Computational time : 7.7 hours.



(b) Four bars (four caustic patterns). Computational time : 15.7 hours.



(c) Rotating Star (nine caustic patterns). Computational time : 27.6 hours.

Figure 10: More results. Note that the caustic pattern set in (c) contain similar patterns, as they are frames of a simple animation.

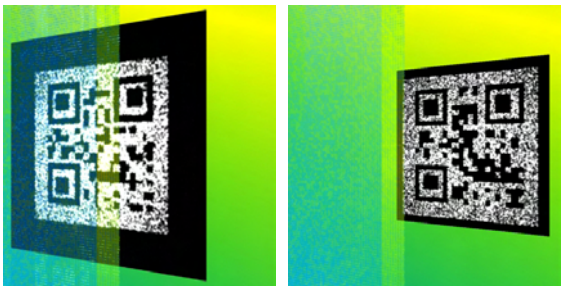


Figure 11: An application of information encoding. We encode **WSCG** and **2012** into two QR barcode patterns.

is also interesting to consider more complex light situations such as area light sources and dynamic light sources. It is challenging to use area light sources since they emit light to many directions from every point in the area light sources. As for the dynamic light sources, it is interesting to generate unique caustic pattern for each given light source direction (in this case, caustic object and caustic receiver are static). Last but not least, we would like to fabricate a real caustic object based on the computed geometry.

9. ACKNOWLEDGEMENTS

This work has been partially supported by the National Research Foundation grant, which is administered by the Media Development Authority Interactive Digital Media Programme Office, MDA (IDMPO). We would like to express our gratitude to Stanford Computer Graphics Laboratory for the 3D models (bunny, and armadillo) which are used to generate the input caustic patterns.

10. REFERENCES

[Ans08a] Anson, O., Seron, F.J., and Gutierrez, D.. NURBS-Based inverse reflector design. in CEIG08: Congreso Español de Informática Gráfica, Eurographics Association, Barcelona, Spain2008. pp. 65–74.

[Bot01a] Bottino, A., Cavallero, L., and Laurentini, A.. Interactive reconstruction of 3-D objects from silhouettes. in WSCG. 2001. pp. 230–236.

[Fin10a] Finckh, M., Dammertz, H., and Lensch, H.P.A.. Geometry construction from caustic images. in Proceedings of the 11th European Conference on Computer Vision: Part V, Springer-Verlag, Berlin, Heidelberg2010. ECCV'10. pp. 464–477.

[Kir83a] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.. Optimization by simulated annealing. Science. vol. 220, no. 4598, pp. 671–680, 1983.

[Mas09a] Mas, A., Martín, I., and Patow, G.. Fast inverse reflector design (FIRD). Computer Graphics Forum. vol. 28, no. 8, pp. 2046–2056, 2009.

[men12a] mental ray. http://www.mentalimages.com/products/mental-ray.html, 2012.

[Mit09a] Mitra, N.J., and Pauly, M.. Shadow art. ACM Trans. Graph.. vol. 28, pp. 156:1–156:7, 2009.

[Pap11a] Papas, M., Jarosz, W., Jakob, W., Rusinkiewicz, S., Matusik, W., and Weyrich, T.. Goal-based caustics. Computer Graphics Forum. vol. 30, no. 2, pp. 503–511, 2011.

[Pat04a] Patow, G., Pueyo, X., and Vinacua, A.. Reflector design from radiance distributions. World Scientific. vol. 10, no. 2, pp. 211–235, 2004.

[Pat07a] Patow, G., Pueyo, X., and Vinacua, A.. User-guided inverse reflector design. Computers & Graphics. vol. 31, no. 3, pp. 501–515, 2007.

[vN51a] von Neumann, J.. Various techniques used in connection with random digits. monte carlo methods. vol. 12, pp. 36–38, 1951.

[Wey09a] Weyrich, T., Peers, P., Matusik, W., and Rusinkiewicz, S.. Fabricating microgeometry for custom surface reflectance. ACM Trans. Graph.. vol. 28, pp. 32:1–32:6, 2009.

[Yue12a] Yue, Y., Iwasaki, K., Chen, B., Dobashi, Y., and Nishita, T.. Pixel art with refracted light by rearrangeable sticks. Computer Graphics Forum. vol. 31, no. 2, pp. 575–582, 2012.

A. NORMAL COMPUTATION

Given the Snell's Equation

$$\begin{aligned} \eta_1 \sin\theta_1 &= \eta_2 \sin\theta_2, \\ \eta\sqrt{1-\cos^2\theta_1} &= \sqrt{1-\cos^2\theta_2}, \\ \eta\sqrt{1-(-\mathbf{N}\cdot\mathbf{M})^2} &= \sqrt{1-(\mathbf{N}\cdot\mathbf{R})^2}, \\ A_1N_xN_x + A_2N_xN_y + A_3N_yN_y &= 1, \end{aligned} \tag{5}$$

with η_1 is the index of refraction of the caustic object, η_2 is the index of the refraction of air, $\mathbf{N}$ is the normal, $\mathbf{M}$ is the inverse incoming light direction, $\mathbf{R}$ is the refracted light direction, θ_1 is the angle between $\mathbf{M}$ and the inverse normal, θ_2 is the angle between $\mathbf{M}$ and the normal, and

$$\eta = \frac{\eta_1}{\eta_2} \qquad A_1 = \frac{\eta^2 M_x M_x - R_x R_x}{\eta^2 - 1}$$
$$A_2 = 2\frac{\eta^2 M_x M_y - R_x R_y}{\eta^2 - 1} \qquad A_3 = \frac{\eta^2 M_y M_y - R_y R_y}{\eta^2 - 1}$$

Since the normal vector $\mathbf{N}$ is normalized, the solution lies on a unit circle and as a result $N_x = \cos\phi$ and $N_y = \sin\phi$. Hence, Equation 5 can be simplified to

$$(2A_3 - 2)\tan^2\phi + 2A_2\tan\phi + (2A_1 - 2) = 0. \tag{6}$$

Equation 6 is basically a quadratic equation and the angle ϕ can be obtained by solving the quadratic equation.

Interactive BRDF Estimation for Mixed-Reality Applications

Martin Knecht
Vienna University of Technology
knecht@cg.tuwien.ac.at

Georg Tanzmeister
Vienna University of Technology
georg.tanzmeister@tuwien.ac.at

Christoph Traxler
VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH
traxler@vrvis.at

Michael Wimmer
Vienna University of Technology
wimmer@cg.tuwien.ac.at

ABSTRACT

Recent methods in augmented reality allow simulating mutual light interactions between real and virtual objects. These methods are able to embed virtual objects in a more sophisticated way than previous methods. However, their main drawback is that they need a virtual representation of the real scene to be augmented in the form of geometry and material properties. In the past, this representation had to be modeled in advance, which is very time consuming and only allows for static scenes.
We propose a method that reconstructs the surrounding environment and estimates its Bidirectional Reflectance Distribution Function (BRDF) properties at runtime without any preprocessing. By using the Microsoft Kinect sensor and an optimized hybrid CPU & GPU-based BRDF estimation method, we are able to achieve interactive frame rates. The proposed method was integrated into a differential instant radiosity rendering system to demonstrate its feasibility.

Keywords

BRDF estimation, reconstruction, augmented reality

1 INTRODUCTION

Many mixed-reality applications require or at least desire a consistent shading between virtual and real objects. Examples are product presentations, virtual prototyping, architectural and urban visualizations and edutainment systems. Here virtual objects should smoothly blend into the real environment and provide a plausible illusion for users. They need to be rendered in a way that makes them hard to distinguish from real objects. Some recently published methods [14, 7] consider the mutual light interaction between real and virtual objects, so that they indirectly illuminate or shadow each other.

Beside the geometry of the scene and the real lighting conditions, the BRDFs of real objects are needed to simulate these mutual shading effects. Acquiring this data in a pre-processing step would diminish the dynamic and interactive nature of mixed-reality systems, and would also make it necessary to track the previously modeled movable real objects. In this paper, we introduce a BRDF estimation method that runs at interactive frame rates. It is based on real-time reconstruction using the structural light scanner provided by Microsoft's Kinect sensor [11]. The real lighting conditions are captured by a camera with a fish-eye lens from which light sources are derived.

Our contribution is best characterized by the unique features of our BRDF estimation approach, which are:

- It runs at interactive frame rates.
- It does not need any pre-processing.
- It utilizes a novel K-Means implementation executed on the GPU.

2 RELATED WORK

BRDF estimation has a long history of research and a variety of methods have been presented. Our approach belongs to the class of image-based methods, which are sometimes synonymously called *Inverse Rendering*. These methods try to fit parameters of an underlying, sometimes rather simple, BRDF model, like

the Phong [15] or Ward model [20], from images of a scene. Yu et al. introduced *Inverse Global Illumination* [23], where reflectance properties are derived from a sparse set of HDR images considering also indirect illumination. The geometry is pre-modeled and partitioned into surfaces with similar materials. The direct light sources must also be known. An optimization algorithm then calculates diffuse and specular components separately. Although the concept is sound and forms the basis of newer algorithms, it needs a lot of manual pre-processing. Sato et al. [17] presented a method that also performs a reconstruction of the object's geometry from range images, which is then used to estimate diffuse and specular parameters from the same images.

Boivin and Gagalowicz [2] use a single LDR image in addition to a geometric model including light sources. Starting with a Lambertian model, they iteratively compare renderings with the original image and consider more and more complex reflectance models as long as the difference is too large. Though their solution is scalable with regard to accuracy, it is still time consuming and requires pre-processing. Mercier et al. [10] were the first to present a fully automatic method to recover the shape and reflectance properties of a single object and the position of light sources from a set of calibrated images. For that purpose, the object and light sources are fixed on a turntable, and photographs are taken every 5 degrees. The geometry is approximated by *Shape From Silhouette* (SFS) from Szeliski [19]. The method is very accurate and does not need any preprocessing, but the special setup makes it unsuitable for mixed-reality. Xu and Wallace [22] used a depth sensor and a stereo intensity image to acquire an object's reflectance properties and parameters for multiple light sources. Although using a depth map comes close to our approach, their method is restricted to a single object. Furthermore, calculating light source parameters from intensity images introduces inaccuracies for flat surfaces.

Zheng et al. [25] presented a solution that is similar to that of Mercier et al. [10]. One big difference is that they use measured lighting conditions instead of deriving this information from the images, which minimizes the estimation error. They then apply the highlight removal algorithm from Ortiz and Torres [13] before clustering images into regions with similar diffuse materials using K-Means. The parameters of the Ward model are then obtained for each cluster by non-linear optimization. Their algorithm is very robust, since after estimating specular factors, diffuse factors are re-estimated in order to compensate for errors caused by wrong clustering or inaccurate geometry.

Like Mercier's method, the approach is based on a controlled setup, which does not meet our requirements. This especially concerns reconstruction by SFS and measurement of the light source. Their estimation pipeline however is very efficient and so we based our work on it. For example we also use an adaptation of the highlight removal technique from Ortiz and Torres [13] and we also use K-Means [9] for clustering.

Several efficient implementations of the K-Means algorithm on the GPU already exist. Almost all of them use a hybrid GPU/CPU approach, where the new cluster centers in each iteration are either entirely or at least partially calculated on the CPU [5, 8, 24, 21]. In all of the aforementioned papers CUDA is used to perform the calculations on the GPU.

To our knowledge there is only one method which was proposed by Dhanasekaran and Rubin [4] where the whole K-Means algorithm is done entirely on the GPU eliminating the need of continuously copying data via the PCIe bus. However, in contrast to Dhanasekaran and Rubin's work which relies on OpenCL, we use a different approach that utilizes mipmaps to calculate the center of each cluster using DirectX.

Generally speaking, all these previous image-based BRDF estimation methods work off-line and have running times ranging from a couple of minutes to several hours. Furthermore they are restricted to static scenes. Mixed-reality applications are highly interactive and dynamic according to Azuma's definition [1]. Hence our motivation was to design and develop a method that runs at interactive frame rates and can thus handle highly dynamic scenes.

3 OVERVIEW

Estimating material characteristics for mixed-reality applications is a challenging task, due to several constraints. On top of it, is the time constraint, since the applications have to be interactive. Then the observered scenes usually exhibit a certain degree of dynamics and materials that just appeared in the camera frustum need to be estimated immediately. As described in the introduction several methods for BRDF estimation exist but all of them are designed for offline purposes. They all try to get a very accurate BRDF estimation. In our case this goal must be lowered to achieve interactive frame rates. The resulting diffuse and specular reflectance maps are used in a differential instant radiosity (DIR) system where the goal is to get visually plausible images instead of physically correct ones. Mapping this idea to our BRDF estimation method, our goal is to find BRDFs that emphasize the same visual cues to the user as the real materials would do.

Our BRDF estimation algorithm is mainly influenced by the ideas of Zheng et al. [25]. Their method was designed to work offline and had thus different requirements. As an adaption we modified their approach where necessary and made extensive use of the GPU

to gain interactive frame rates. The presented method can be divided into two main parts:

- Data acquisition: Capture data from the Microsoft Kinect sensor and a fish-eye lens camera to obtain color, geometry and lighting information.
- BRDF Estimation: Enhance the RGB input data and estimate diffuse and specular material characteristics.

Figure 1 illustrates the separate steps in the context of the two main parts of the method. Section 4 describes the data acquisition. Here normals are obtained with the use of the depth map we get from the Kinect and the lighting environment is approximated with the input stream of the fish-eye lens camera. The BRDF estimation is described in Section 5 where after intermediate steps the final diffuse and specular reflectance parameters are estimated. The output of our method is integrated into a DIR rendering system [7] and we directly render the fully defined geometry (including material characteristics) into a G-Buffer which stores the 3D position, the normal, the color and the material parameters needed for Phong shading.

4 DATA ACQUISITION

The Microsoft Kinect sensor is a relatively cheap device to capture a video and a depth stream simultaneously. The resolution of both streams is 640×480 pixels at a frame rate of 30Hz. Surrounding objects can be captured in a range between 0.45 and 10 meters. Figure 2 shows the input data provided by the Kinect sensor. The other source of input data is a IDS uEye camera with a fish-eye lens attached to capture the incident illumination.

4.1 Normal Estimation

We implemented two methods for normal estimation. The first one uses the Point Cloud Library (PCL) [16]. While the normals are of high quality, their computation takes too much time. The PCL functions need 223 milliseconds for one normal estimation step with a smoothing factor of 10. The estimation is performed on the CPU and therefore we implemented our own GPU normal estimation method that exploits temporal coherence (TC) between adjacent frames in a similar way as done by Scherzer et al. [18].

Our normal estimation is based on two render passes. The first pass performs subsampling and averaging of the normals from the previous frame. Furthermore, a curvature coefficient is calculated. The subsampling causes a smoothing on the normals of the previous frame. Let (i, j) be the row and the column of a given pixel in the previous frame. The average normal is

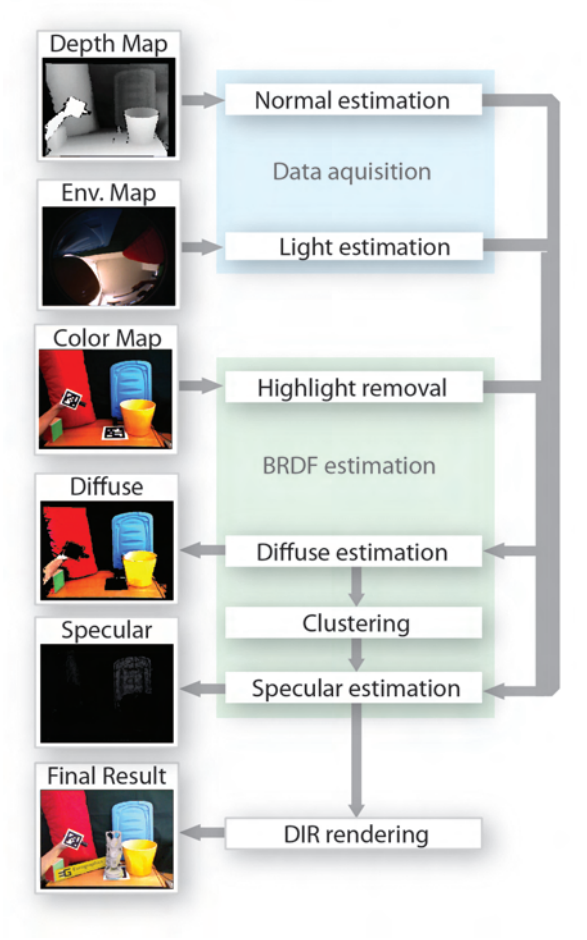


Figure 1: Shows the main steps in the BRDF estimation pipeline. Operations related to data acquisition are shown in the blue box (Section 4). Steps belonging to BRDF estimation and are shown in the green box (Section 5).

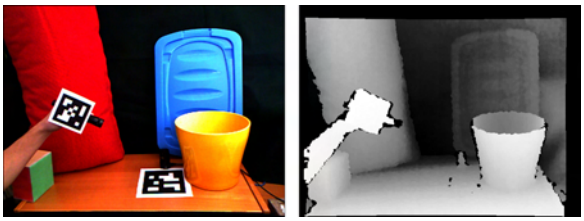


Figure 2: The left image shows the video input stream. The right image shows the normalized depth input stream. Both streams have a resolution of 640×480 with a frame rate of 30Hz.

then calculated by averaging over $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ and $(i, j+1)$. Note that if no normal is available at a given pixel location, it will be discarded from the calculation. The curvature coefficient is calculated as follows:

$$curv_H(i,j) = N_{i-1,j} \cdot N_{i+1,j} \quad (1)$$

$$curv_V(i,j) = N_{i,j-1} \cdot N_{i,j+1} \quad (2)$$

$$curv(i,j) = \min\left[curv_H(i,j), curv_V(i,j)\right]^{128} \quad (3)$$

where the dot is the dot product operator. Note that the curvature coefficient goes to zero at sharp edges and to one at flat areas. The average normal and the curvature coefficient of the last frame are rendered to a render target with half the dimension of the rendering window.

The second rendering pass consists of two steps. In the first one a new normal is calculated from the point cloud delivered by the Microsoft Kinect sensor. We look up the 3D position $p_{i,j}$ at the current pixel (i,j) and two neighboring positions in horizontal $(i,j+4)$ and vertical $(i+4,j)$ direction. A distance value of four pixels showed good smoothing characteristics while edges were still preseverd. From these values, we can set up a surface normal as follows:

$$d_{i+4,j} = \frac{p_{i+4,j} - p_{i,j}}{|p_{i+4,j} - p_{i,j}|} \quad (4)$$

$$d_{i,j+4} = \frac{p_{i,j+4} - p_{i,j}}{|p_{i,j+4} - p_{i,j}|} \quad (5)$$

$$normal_{i,j} = d_{i+4,j} \times d_{i,j+4} \quad (6)$$

In the second step the information calculated by the first rendering pass is used to calculate an old average normal. First the lookup coordinates are calculated by using reprojection. In this way the camera movement from one frame to another can be canceled out. The curvature coefficient at the current pixel steers the mipmap level for the lookup of the previous normal. The new and the previous normal vectors are linearly combined depending on a confidence value calculated as follows:

$$c_N = |N_p \cdot N| \quad (7)$$

$$c = c_B * c_N + (1 - c_N) \quad (8)$$

where N_p is the previous averaged normal and N is the new normal. c_N is the confidence coefficient based on the similarity of the previous and the new normal. The resulting confidence is a linear blend between a base confidence c_B and 1, steered by c_N. To deal with disocclusions occurring during camera movement, we set the confidence value c to zero if the depth difference between the old frame and the new frame is larger than 0.1 meters. In this way, normals at dynamic elements get updated faster.

While the quality of the normals is not that high compared to the results of the PCL, our proposed method runs on the GPU and is thus faster (see Section 6). Furthermore note that the reprojection quality heavily depends on the tracking quality.

4.2 Light Estimation

The incoming light position must be known in order to be able to estimate a BRDF. The fish-eye camera captures the environment map and the DIR rendering system creates a dome of virtual point light (VPL) sources above the scene. We select a subset of these VPLs from the dome and use them for BRDF estimation. The selection criteria are that the VPLs have a high intensity and that there is no other selected VPL within certain distance.

5 BRDF ESTIMATION

Similar to Zheng et al. [25] highlights in the input image are removed and afterwards inverse diffuse shading is applied. However, in their approach the resulting buffer was just used for clustering. In contrast we also use this buffer as a diffuse reflectance map to keep the computation time low.

5.1 Highlight Removal

To estimate specular reflectance values similar colors need to be clustered since they are assumed to belong to the same material. However, specular highlights would form a separate cluster due to saturation, which is not desired. Our highlight removal is based on the work of Ortiz and Torres [13]. Instead of transforming the camera color image into the L1-Norm, we use the Hue Saturation Intensity (HSI) color space. Highlights should be detected at pixels where the color has high brightness but low saturation. As thresholds we set the minimum brightness to 0.9 and the maximum saturation to 0.1. In a first pass, the highlight detection result is written into a binary mask with a one where the brightness and saturation criteria are met and a zero otherwise. Then a morphological dilation with a disk (radius of 4 pixels) is performed. While Ortiz and Torres [13] perform a *Morphological Vectorial Opening by Reconstruction*, we use a rather simplistic reconstruction method. For each pixel that is masked as a highlight, a new color has to be found that ideally matches surrounding colors. We do this by iterating through neighboring pixels in an increasing circular manner until a pixel is found that is not masked as belonging to a highlight anymore. Then the color of the found pixel is used to substitute the color of the masked pixel. In this way, all highlights can be canceled out. Note that due to this highlight removal process, bright and weakly saturated objects may get misinterpreted as highlights. The results of the highlight removal operation are shown in Figure 3.

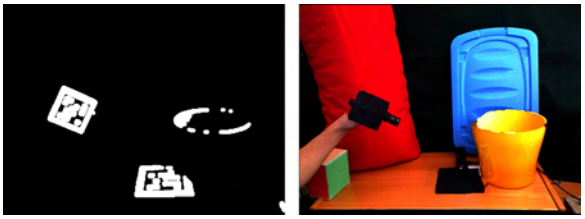


Figure 3: The left image shows the highlight mask. In a second step the masked pixels are filled as shown in the image on the right.

5.2 Diffuse Reflectance Estimation

After highlight removal we estimate the diffuse parameters per pixel by rephrasing the diffuse illumination equation. We end up with the following formula for the diffuse reflectance estimation k_d:

$$k_d = \frac{I}{\sum_{l=1}^{n} I_l (N \cdot L_l)}, \tag{9}$$

where I is the input intensity of the current pixel, I_l the intensity of the lth light source, L_l the direction towards the lth light source and n is the number of lights that are used for the BRDF estimation. Zheng et al. [25] estimate the diffuse parameters at a later stage because they used multiple RGB samples per vertex. We use the resulting buffer as diffuse reflectance map and as input for the clustering. The estimated diffuse reflectance map is shown in Figure 4. In the ideal case the different objects would have completely flat colors. However, this is not the case due to several simplifications that introduce consecutive errors in the pipeline. First, the environmental light is represented by only a few virtual point lights. Second, no shadows or indirect illumination are taken into account and third, the normal estimation is not absolutely correct.

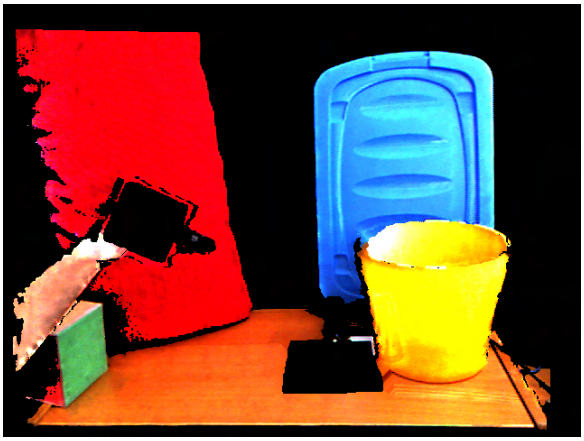


Figure 4: This image shows the estimated diffuse material component k_d. In the ideal case the objects would look perfectly flat and no variations due to different normals and thus illumination would be visible.

5.3 Clustering

Pixels with similar RGB colors in the diffuse reflectance map are assumed to have the same material and therefore need to be clustered. A novel K-Means implementation that is executed on the GPU performs the clustering. K-Means was introduced by Stuart P. Lloyd [9] and it consists of the following steps:

1. Randomly choose k cluster centers.
2. Assign each data element to the nearest cluster center using Euclidean distance.
3. Calculate new cluster centers by calculating the centroid over all data elements assigned to a specific cluster.
4. Repeat steps 2 & 3 until termination criteria are met.

Step 1: Initialize cluster centers: The resulting clusters heavily depend on the initial values chosen for the cluster centers. Thus if bad initial cluster centers are chosen, it might take many iterations until convergence. For each frame, we therefore use one to two different initial cluster centers. The first set uses the cluster centers from the previous frame and if the stopping criteria are met (see step 4) the next iteration is not executed anymore. However, if they are not met, the second set is executed with random cluster center values.

Step 2: Assign element to nearest cluster: Step two is adapted slightly so that step 3 can be executed on the GPU. Instead of just outputting the nearest cluster id and the minimum distance, we need to render each color pixel into multiple render targets. The idea is that each cluster has its own render target and pixels rendered into a given render target only belong to a certain cluster. We used eight simultaneous render targets and can handle six clusters each time a screen space pass gets executed. The following information is stored on a per-cluster basis for each pixel:

- The *RGB* color value
- The minimum distance to the nearest cluster center
- A binary flag that defines to which cluster the pixel belongs

The *RGB* color and minimum distance can be stored in one texture buffer with four floating point values. For the binary flags of all six clusters, we used two textures where every cluster gets assigned to one color channel. Depending on the cluster id assigned to a given pixel color, the color and distance information is only written into the textures assigned to the specific cluster. All other render target values are set to zero.

Step 3: Calculate new cluster centers: In step three we need to calculate the average *RGB* value for each cluster which is then used as a new cluster center. For a texture T with a size of $2^n \times 2^n$, there are n mipmap levels that can be created. The smallest mimap level with a size of 1×1 stores the average value of all data in texture T. However, we only want the average *RGB* color of those pixels that belong to a given cluster and ignore those that were set to zero.The cluster center can therefore be calculated using a combination of the two lowest mipmap levels from the color texture and the binary flag texture as follows:

$$cluster_c(T_{RGBD}, T^*) = \frac{avg(T_{RGBD})}{\sum_{i=0}^{n} \sum_{j=0}^{n} T^*_{i,j}} \tag{10}$$

where T_{RGBD} is a cluster specific texture containing the *RGB* color values and the distance value. T^* is the binary texture for the cluster having ones where pixels are assigned to that cluster and zeros otherwise.

Step 4: Repeat steps 2 & 3: In the original K-means method [9], the second and third steps are repeated until no data element changes the cluster anymore. This stopping criteria is too conservative for our needs. We need a fast clustering algorithm and thus have lowered the stopping criteria: First only a maximum number of 20 iterations are performed defining the upper bound for the computation time. Second if the variance change from one iteration to another drops below 10^{-4}, no further iterations are executed. By exploiting temporal coherence a low variance solution may be available after the first iteration and no new cluster center set needs to be processed. Note that the variance is calculated in a similar way to the cluster centers by exploiting mipmapping. As the squared distances for each pixel to the cluster centers are already calculated in the shader, the variance can be calculated nearly for free in step 2.

If the first run with the old cluster centers as initial values does not converge, the second run with random cluster centers gets executed. Then the cluster centers with the lower variance value are used for BRDF estimation. However, always using just the previous cluster centers could lead to a local minimum for clustering and there would be no way out to maybe find the global one. For this reason, in every fifth frame, the second iteration with random cluster centers will be executed anyway. Figure 5 shows the resulting clusters after K-Means is applied on the diffuse reflectance map.

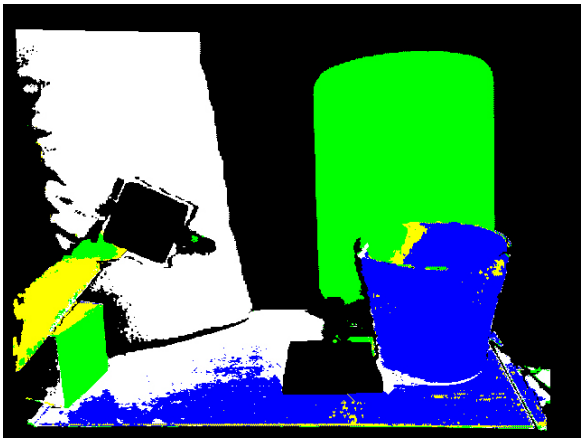


Figure 5: This figure shows the resulting clusters after K-Means is applied.

5.4 Specular Reflectance Estimation

One of the last steps needed in the BRDF estimation pipeline is the estimation of the specular intensity k_s and specular power n_s values per cluster. We assume white highlights and thus k_s is reduced to a scalar value. The paramters are estimated similar as proposed by Zheng et al. [25]. However, there are two main differences in our method. First, the solver works partly on the GPU and thus gains more speed than just a plain CPU implementation. Second, the positions of the light sources are not chosen to be fixed variables. The reason for this is that the positions are evaluated using importance sampling and thus can vary over time and furthermore need not to be at the exact position where a small light source is placed. However, the position of a light source highly influences the position of the specular reflection and therefore small variations of the initial positions are allowed to the solver.

For the non-linear optimization, a Nelder-Mead algorithm was used [12] with the following objective function evaluated on the GPU:

$$F_j = \sum_i \left[I_i - \sum_{l=1}^{n} I_l k_d (N \cdot L_l) + I_l k_s (V \cdot R_l)^{n_s} \right]^2 \quad (11)$$

where i iterates over all pixel intensities I_i which are related to cluster j. I_l is the intensity of the lth light source and k_d is the diffuse intensity vector of a cluster, which is set to the cluster center color. Note that for the specular component estimation, k_d is fixed and only the light source positions as well as k_s and n_s can be varied by the solver. N is the normal vector of the surface and R_l the reflection vector of the lth light source. V is a view vector pointing towards the camera. The result of the specular reflectance estimation is shown in Figure 6. Figure 7 shows a simple Phong-illuminated rendering on the left using the estimated k_d, k_s and n_s Phong illumination parameters. In this case, the same VPLs are used for illumination that are also used to estimate the BRDFs. In the image on the right side a rendering using DIR with an additional virtual pocket lamp is shown. Note the yellow indirect illumination on the real desk and on the virtual Buddha.

Figure 6: This image shows the result of the specular component estimation.

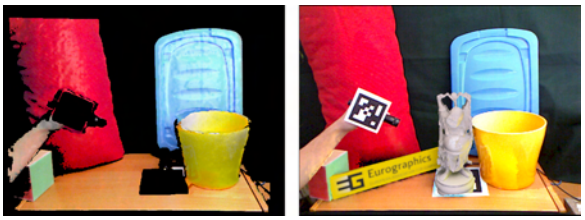


Figure 7: The left image shows a simple Phong rendering with the VPLs used for BRDF estimation. In the right image a virtual pocket lamp illuminates the real scene. Note the yellow color bleeding on the real desk and on the virtual Buddha.

6 RESULTS

Our computer system consists of an Intel Core 2 Quad CPU at 2.83 GHz and an NVIDIA GeForce GTX 580 with 1.5 GB of dedicated video memory. The software is running on Windows 7 64-bit and implemented in C# and DirectX 10. Cameras used in this scenario are the Microsoft Kinect sensor and an IDS uEye camera to capture the environment map.

6.1 Normal estimation

The two methods used for normal estimation differ in quality and computation time. Figure 8 shows a side-by-side comparison of the normal maps. The left normal map is calculated with the PCL library and a smoothing factor of 10. The average computation time is 223 milliseconds. The right normal map is computed with our proposed method in about 0.57 milliseconds. The PCL based normal map has a lot of holes, shown in grey color. In our method these holes are filled with the normals of the neighboring pixels. Even though these normals are not correct from a reconstruction point of view, they reduce the visible artifacts a lot. Furthermore note that our method produces sharper edges.

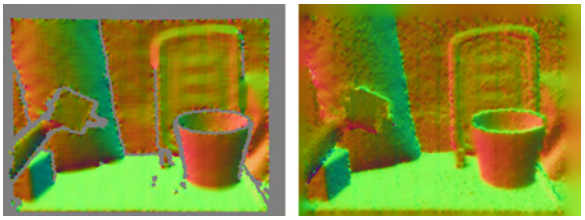


Figure 8: Comparison of the two implemented normal estimation methods. Left: Normals estimated using the PCL library [16] in 223 milliseconds. In grey areas no normal could be estimated by the PCL. Right: Our proposed method which takes 0.57 milliseconds.

6.2 K-Means clustering

We compared the proposed K-Means clustering implementation against the OpenCV library [3]. In the test setup a data set of 640×480 3-vector elements needed to be clustered. We ran both algorithms 5 times each time with different initial cluster centers. The interaction count of each run was set to 20. Note that no temporal coherence was exploited in order to get comparable results. The measured times include all the 5 runs and do not include the setup of the random data elements. Table 1 shows the execution times in seconds for 6 and 12 clusters.

Clusters	OpenCV	GPU K-Means
6	3.94s	0.33s
12	7.07s	0.44s

Table 1: Shows a comparison between the K-Means implementation from OpenCV [3] and our GPU implementation. Both algorithms ran 5 times with 20 iterations. Timings show the total execution of the 5 runs in seconds.

Table 2 shows the average iteration count needed for the first 50 frames to get below the variance change threshold. The columns show the average number of iterations for 6 clusters (6C) and for 12 clusters (12C). The rows show if the cluster centers from the previous frame were used (frame^{-1}) or if the cluster centers where chosen randomly (random). We set the maximum iteration count to 30, which was never reached during this test.

Initials	Avg. Iter., 6C	Avg. Iter., 12C
random	9.00	11.47
frame^{-1}	7.53	6.98

Table 2: Shows the average iteration count when reusing the cluster centers from the previous frame or taking random new ones.

6.3 Performance Analysis

The BRDF estimation pipeline has several steps. Table 3 gives an overview on the time spent for each one. In this setup, 6 clusters and 4 VPLs were used to approximate the BRDFs.

Stage	Time in ms
Normal Estimation	0.57ms
Highlight Removal	0.94ms
Diffuse estimation	0.23ms
K-Means	39.08ms
Specular estimation	315.76ms
Total time:	356.58ms

Table 3: Shows the time spent on each pipeline stage.

It clearly shows that the specular estimation step consumes by far most of the time. However, if it is possible not only to use a hybrid CPU / GPU version for the optimization but a complete GPU solution, the performance should increase a lot.

Two main parameters can be tweaked to get a better performance for a given scenario. One parameter is the number of materials that are estimated every frame. The

second parameter is the number of virtual point lights that are used to approximate the surrounding illumination. Table 4 shows the impact of different cluster and VPL settings with a fixed maximum iteration count for the specular estimation set to 50.

VPLs	6 Cluster (fps)	12 Cluster (fps)
1	3.82	2.41
8	2.23	1.30
128	1.70	1.01
256	0.99	0.59

Table 4: Shows the average fps with different VPL and cluster settings.

We also investigated how the maximum iteration count for the specular estimation solver reduces the total error. Interestingly, the change of the error was extremely small regardless how large the value was set. We think that this has to do with the large amount of pixels that are available in a cluster. Compared to that the area of a specular highlight is relatively small and thus correct estimations will only have a small impact on the total error.

Furthermore it turned out that the solver has difficulties in finding appropriate values in certain cases. Sometimes there is simply no highlight due to a given VPL. We therefore introduced a threshold value for a maximum error. If the error is too large, we set the specular intensity k_s to zero. Another problem could be that the solver just has one single point of view per frame whereas Zheng et al. [25] used several photographs to perform a specular estimation. Recently upcoming techniques, however, promise to greatly improve the problem of temporal coherent BRDF estimation (see Section 8).

6.4 Critical Scenario

A critical scenario is shown in Figure 9 on the left. It shows a blue notebook and a horse made from porcelain placed on a wooden box. The light is mainly coming from the direction of the spotlight partially visible on the left side of the image, causing specular highlights on the blue notebook envelope and the wooden box. The number of clusters used in this scenario is six, and four virtual point lights are used to estimate the surrounding illumination. The average frame rate is 2.3 fps.

In this scenario, the clustering is not as distinct regarding the objects compared to the first scenario. Due to the highlights caused by the spotlight, the clustering step creates different clusters for the same material as shown in Figure 9 (right). Furthermore, the Kinect sensor has troubles finding depth values at the white borders of the tracking marker, resulting in holes in the estimations (see Figure 10 (left)). Figure 11 shows the resulting diffuse (left) and specular (right) estimations. The Phong-shaded result is shown in Figure 12.

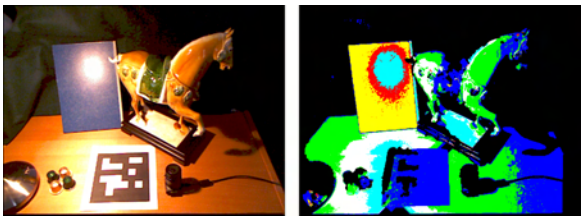


Figure 9: The left image shows the video input captured by the Kinect sensor. On the right side, the clustering result is shown.

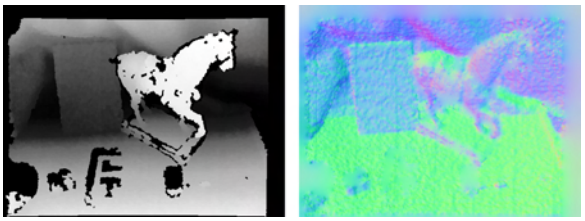


Figure 10: This figure shows the depth values acquired by the Kinect sensor on the left. Note that it failed to measure depth at the white borders of the tracking marker and the black fish-eye lense camera. On the right side the normal estimation from our proposed method is shown.

Figure 11: This figure shows the scene rendered with just the diffuse estimation (left) and specular estimation (right).

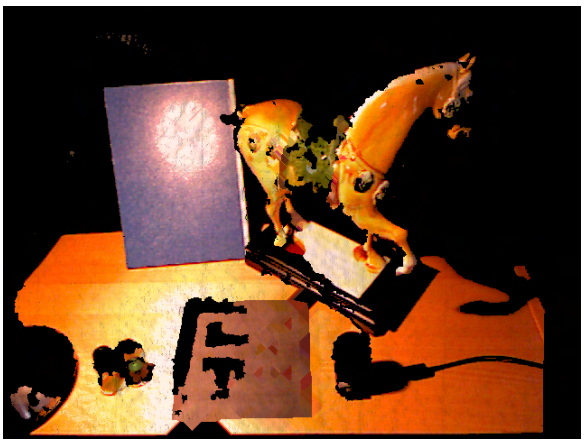


Figure 12: This figure shows the Phong-shaded result combining the diffuse and specular estimations.

7 LIMITATIONS AND DISCUSSION

Some limitations of our method are imposed by the Microsoft Kinect sensor, which is a structural light scanner. In general, depth values cannot be calculated when the light pattern is not recognized by the system. This

happens when objects are very glossy, reflective, transparent or absorb too much of the infrared light. The infrared pattern also vanishes in direct sunlight, making the approach unsuitable for outdoor mixed-reality. Furthermore, the border of curved objects is also often missing from the depth map because the projected light pattern is too distorted there.

Since bright pixels are assumed to be highlights due to specular reflections, bright and weakly saturated objects may be misinterpreted as highlights. Furthermore, shadows are not considered directly in the current implementation. Pixels with a brightness value below a certain threshold are simply discarded.

The K-Means clustering approach uses a variance value to decide whether further iterations are needed or not. However, there is no estimation of the optimal amount of clusters right now. This number must be specified by the user in advance and highly depends on the materials available in the scene.

Although temporal coherence is exploited at several stages in the pipeline, we do not continuously integrate already-seen geometry data. This would be helpful as a given point in the scene could be viewed under different viewing angles, leading to a better BRDF estimation, but could also lead to problems with moving objects.

Due to the real-time constraints several simplifications are introduced. The environmental illumination is approximated using a few virtual point lights, the normals have a lower quality compared to the PCL library and the clustering therefore also introduces some errors. All these simplifications lower the quality of the final BRDF estimation. However, since DIR mainly tries to compute visually plausible results rather than being physically correct, the estimated BRDFs should have a sufficient quality for mixed-reality scenarios.

8 CONCLUSION AND FUTURE WORK

We introduced a method to estimate the BRDFs of an augmented scene at interactive frame rates. The method does not need any precomputation, which makes it suitable for mixed-reality applications. The Microsoft Kinect sensor serves as a data input source to reconstruct the surrounding environment in the form of geometry and material properties. First, normals are estimated using a screen-space method exploiting temporal coherence. In the next pipeline stage we propose an adapted K-Means implementation that is specially tailored towards BRDF estimation and fast execution on the GPU. Temporal coherence is exploited here too, which allows us to find clusters faster than with a conventional implementation. The Phong parameter estimation is performed using a hybrid CPU / GPU variation of the Nelder-Mead optimization algorithm. The results demonstrate the feasibility of this method for mixed-reality applications.

In the future, we plan to enhance the quality and speed of this BRDF estimation method. It should be possible to gain a lot of speed by porting the Nelder-Mead optimization method to the GPU. Furthermore, recent techniques like KinectFusion [6] could greatly enhance the quality of the BRDF estimation.

9 ACKNOWLEDGEMENTS

This work was supported by a grant from the FFG-Austrian Research Promotion Agency under the program “FIT-IT Visual Computing” (project no. 820916). Studierstube Tracker is kindly provided by Imagination Computer Services GmbH.

10 REFERENCES

[1] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[2] Samuel Boivin and Andre Gagalowicz. Image-based rendering of diffuse, specular and glossy surfaces from a single image. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH ’01, pages 107–116, New York, NY, USA, 2001. ACM.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.

[4] Balaji Dhanasekaran and Norman Rubin. A new method for gpu based irregular reductions and its application to k-means clustering. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-4, pages 2:1–2:8, New York, NY, USA, 2011. ACM.

[5] Bai Hong-tao, He Li-li, Ouyang Dan-tong, Li Zhan-shan, and Li He. K-means on commodity gpus with cuda. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 03*, CSIE ’09, pages 651–655, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST ’11, pages 559–568, New York, NY, USA, 2011. ACM.

[7] Martin Knecht, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential instant radiosity for mixed reality. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '10, pages 99–108, 2010.

[8] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu. Speeding up k-means algorithm by gpus. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 115–122, Washington, DC, USA, 2010. IEEE Computer Society.

[9] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[10] Bruno Mercier, Daniel Meneveaux, and Alain Fournier. A framework for automatically recovering object shape, reflectance and light sources from calibrated images. *International Journal of Computer Vision*, 73:77–93, June 2007.

[11] Microsoft. Kinect sdk: research.microsoft.com/en-us/um/redmond/projects/kinectsdk/.

[12] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[13] F Ortiz and F Torres. Automatic detection and elimination of specular reflectance in color images by means of ms diagram and vector connected filters. *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews*, 36(5):681–687, 2006.

[14] Saulo A. Pessoa, Guilherme de S. Moura, Veronica Teichrieb, and Judith Kelner. Photorealistic rendering for augmented reality: A global illumination and brdf solution. In *Virtual Reality*, pages 3–10, 2010.

[15] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18:311–317, June 1975.

[16] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[17] Yoichi Sato, Mark D. Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 379–387, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[18] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45–50. Eurographics, Eurographics Association, June 2007.

[19] Richard Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Underst.*, 58:23–32, July 1993.

[20] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 265–272, New York, NY, USA, 1992. ACM.

[21] Ren Wu, Bin Zhang, and Meichun Hsu. Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW '09, pages 1–6, New York, NY, USA, 2009. ACM.

[22] S. Xu and A. M. Wallace. Recovering surface reflectance and multiple light locations and intensities from image data. *Pattern Recogn. Lett.*, 29:1639–1647, August 2008.

[23] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: recovering reflectance models of real scenes from photographs. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 215–224, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[24] M. Zechner and M. Granitzer. Accelerating k-means on the graphics processor via cuda. In *Intensive Applications and Services, 2009. INTENSIVE '09. First International Conference on*, pages 7 –15, april 2009.

[25] Zuoyong Zheng, Lizhuang Ma, Zhong Li, and Zhihua Chen. Reconstruction of shape and reflectance properties based on visual hull. In *Proceedings of the 2009 Computer Graphics International Conference*, CGI '09, pages 29–38, New York, NY, USA, 2009. ACM.

A Hierarchical Splitting Scheme to Reveal Insight into Highly Self-Occluded Integral Surfaces

Andrea Brambilla
University of Bergen
Bergen, Norway
andrea.brambilla@uib.no

Ivan Viola
University of Bergen
Bergen, Norway
ivan.viola@uib.no

Helwig Hauser
University of Bergen
Bergen, Norway
helwig.hauser@uib.no

Abstract

In flow visualization, integral surfaces are of particular interest for their ability to describe trajectories of massless particles. In areas of swirling motion, integral surfaces can become very complex and difficult to understand. Taking inspiration from traditional illustration techniques, such as cut-aways and exploded views, we propose a surface analysis tool based on surface splitting and focus+context visualization. Our surface splitting scheme is hierarchical and at every level of the hierarchy the best cut is chosen according to a surface complexity metric. In order to make the interpretation of the resulting pieces straightforward, cuts are always made along isocurves of specific flow attributes. Moreover, a degree of interest can be specified, so that the splitting procedure attempts to unveil the occluded interesting areas. Through practical examples, we show that our approach is able to overcome the lack of understanding originating from structural occlusion.

Keywords: flow visualization, illustrative visualization, occlusion management.

1 INTRODUCTION

Flow phenomena are present at very different scales in our world, and they influence many aspects of our daily life: winds and water currents determine weather and climate, the stream of air around vehicles affects their speed and stability, the flow of blood in our vessels is fundamental for our good health condition. Understanding their behaviour is therefore highly relevant in many fields, and several years of research in *flow visualization* have produced a wide set of tools to accomplish this difficult task [PVH+02].

Flow behaviour can be analyzed from different points of view, according to the specific needs of the user. In particular, field experts are often interested in the trajectories of massless particles that are advected by the flow, which are commonly visualized using *integral curves*. Specifically, a *path line* represents the trajectory of a massless particle seeded from a specific starting location. Similarly, a *path surface* conveys the trajectories of a set of particles seeded along a $1D$ curve.

Integral surfaces are very expressive, but have a major downside: in correspondence with areas of swirling motion, like vortices and eddies, they tend to fold and twist, becoming very intricate and difficult to understand (Figures 2, 7, and 8). In this paper, we present a procedure which aims at solving this issue using techniques from traditional handcrafted illustration, such as cutting and splitting (Figure 1). These concepts have been frequently applied in medical visualization scenarios, but their application in the context of flow visualization has been limited. This is probably due to the fact that identifying well defined objects in flow data is very challenging. An overview of related approaches is presented in Section 2.

We propose a general surface splitting methodology based on two main concepts: a *cut space* defines possible ways to split a surface so that the resulting pieces have a clear meaning, while a *complexity measure* de-

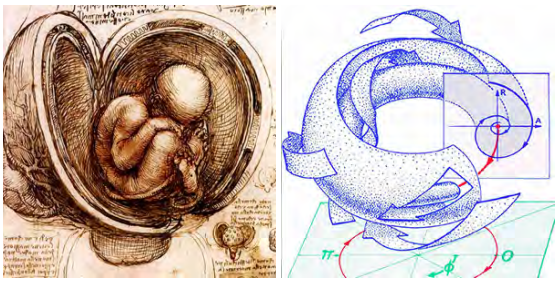


Figure 1: (left) Example of a cut-away view in a traditional illustration by Leonardo da Vinci [dV11]. (right) Illustration of a stream surface with cuts and clipping planes, by Abraham and Shaw [AS82].

termines a degree of occlusion at every point on the surface. We iteratively split the surface according to a cut from the cut space, so that the complexity is reduced the most. To improve the versatility of our approach, we allow the user to specify a degree of interest (DoI) function over the surface, which is combined with the complexity measure when the cut is chosen. Details on the splitting algorithm can be found in Section 3.

The resulting pieces of the surface are presented in a tree-like structure, and pieces of interest can be visualized either separated from the rest of the flow structure, or with a semi-transparent context (Figure 2). We use a stream surface extracted from the ABC flow to illustrate our method. We then show the application of our method on two datasets from application fields. Section 4 describes this process and provides a short discussion on timings and computational complexity.

Compared to the current state of the art, the main contributions of our work are:

- a general methodology for the design of surface cuts
- the first (to the best of our knowledge) splitting approach for integral surfaces
- a novel complexity measure for surfaces, which can take into account the importance of the data
- a helpful tool for the analysis of stream surfaces.

2 RELATED WORK

According to one of the most well-known categorizations [PVH+02], flow visualization techniques can be classified in four groups: direct, texture-based, geometric and feature-based visualization. Our work is related to the third category. Geometric approaches in fact aim at visualizing flow data through *integral structures*. The most common types of $1D$ integral curves are

- *streamlines*: curves tangent to the flow field in every point at a specific time instant
- *path lines*: the trajectories of massless particles in steady or unsteady flows
- *streak lines*: formed by particles continuously released in the velocity field from a specific location
- *time lines*: curves connecting a set of particles simultaneously released along a seeding curve.

These concepts can be extended to $2D$ and $3D$, obtaining surfaces and volumes respectively. Interested readers can refer to to the excellent survey by McLoughlin et al. [MLP+10] for more details.

Flow datasets are often multidimensional, multivariate and very dense. In these cases, traditional flow visualization approaches often suffer from cluttering and occlusion problems, which are commonly addressed with simple techniques, such as clipping, slicing or conventional transparency. A novel visualization research direction, called *illustrative visualization* [RBGV08], aims at solving these perceptual issues taking inspiration from traditional handcrafted illustrations.

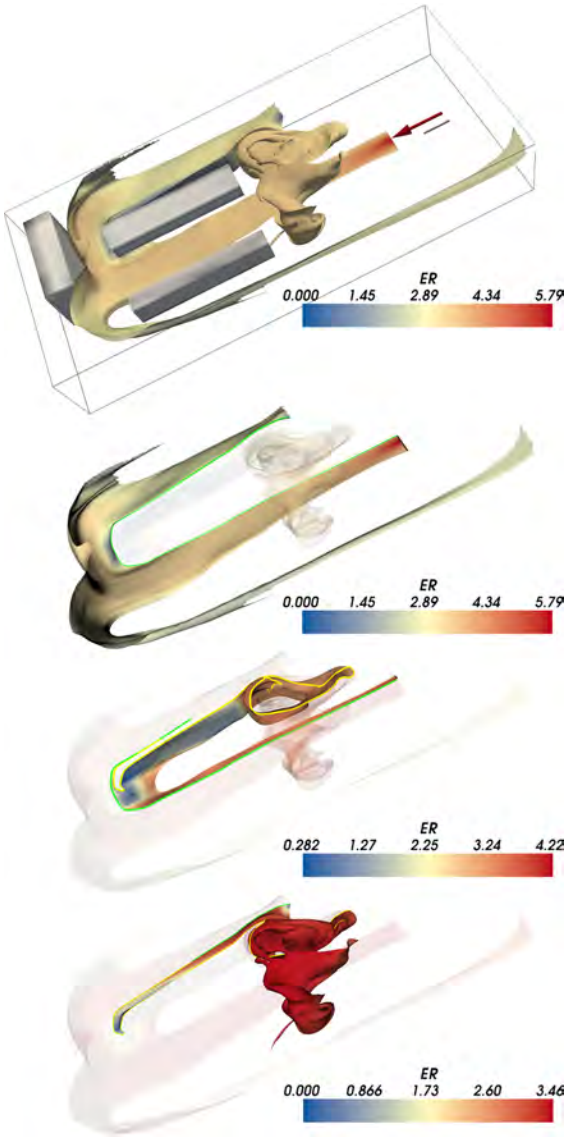


Figure 2: A stream surface extracted from a simulation of a gas leak on an oil platform. Top image: the initial surface with the position of the leak (red arrow) and the objects placed in the room (gray structures). Bottom three images: the surface pieces obtained after two cuts.

Cutting an object to reveal its inner parts is a common approach in illustrative visualization, and it can be applied in different ways. A typical example are *exploded views*: Li et al. [LACS08] apply this concept to show how composite objects are built. Ruiz et al. [RVB+08] suggest to subdivide a volume into oriented slabs according to the amount of information conveyed. More recently, Karpenko et al. [KLMA10] propose an

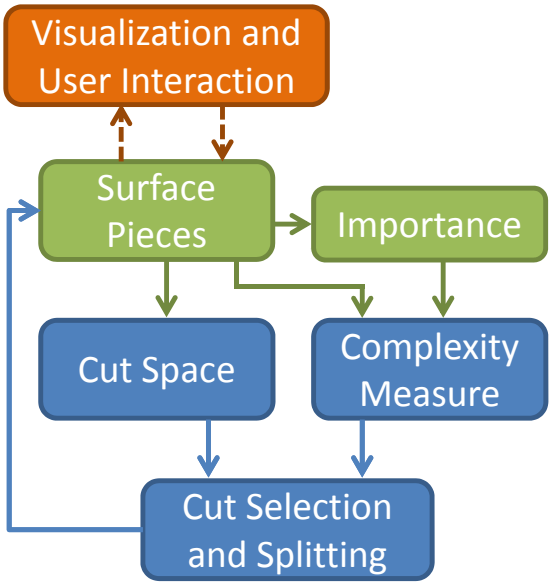


Figure 3: An overview of the splitting algorithm.

explosion strategy for mathematical surfaces based on surface symmetries.

If an importance measure is defined over the data, the visualization could be guided by these values. For instance, Viola et al. [VKG05] describe a volume rendering technique which discards the low-importance (context) portions of the volume occluding the relevant ones (focus). Similarly, Bruckner and Gröller [BG06] propose an exploded view strategy, where the occluding context is not discarded, but displaced in an intuitive way. Bruckner and Gröller also presented a concise overview of basic focus+context approaches in 2005 [BG05]. An effective combination of splitting and focus+context visualization has been presented by Balabanian et al. [BVG10]. Their work is focused on medical volumetric data and the splitting is based on a precomputed segmentation. The resulting pieces are displayed in a navigable graph, which was the main inspiration for our subdivision hierarchy.

Illustrative principles have been mainly adopted in medical visualization, but, especially in recent years, they are spreading to other contexts as well. For flow visualization, a fair number of illustrative techniques have been proposed [BCP+12]. The self-occlusion problem of integral surfaces have been initially addressed in an early paper by Löffelman et al. [LMGP97]: their approach cuts away pieces of the surface, generating results similar to the illustrations by Abraham and Shaw (Figure 1, right).

Two relevant focus+context approaches have been proposed in 2005 and 2007 respectively. The Eyelet particle tracing approach [WS05] shows integral surfaces passing through a specific point of high interest. In contrast, the technique by Correa et al. [CSC07] computes a deformation of the low importance data so that the focus is not occluded. More recently, two noteworthy approaches [HGH+10, BWF+10] propose to address the self-occlusion problem of stream surfaces through a smart use of transparency. They also adopt ad-hoc shading and texturing in order to improve depth perception and convey local flow behaviour.

Outside the context of flow visualization, similar issues have been investigated in connection with isosurfaces of scalar volumes. In this field, many techniques have been proposed (the contour spectrum [BPS97], Reeb graphs [FTAT00] and similarity maps [BM10], just to mention a few), but their applicability to flow data is still uncertain.

3 SURFACE SPLITTING

In the case of $3D$ flow fields, a stream surface is a $2D$ manifold. Our algorithm assumes it is represented by a triangular mesh. The mesh is defined by a set of points $P \subset \mathbb{R}^3$, and a set of triangles T. Flow data is sampled at each point in P: for instance, the velocity at a point $\mathbf{p} \in P$ is $\mathbf{v}(\mathbf{p})$. Linear interpolation is used to determine flow attributes over the triangles.

The structure of our general splitting framework is summarized in Figure 3. The splitting process is iterative and begins when the user requests to generate a cut. At this point two independent steps are performed: the complexity measure $cpx(\cdot)$ is computed for every $\mathbf{p} \in P$ and a set of potential cuts (the cut space) is generated. The complexity measure can take into account a degree of interes $doi(\cdot)$ defined over the points.

Notice that, regardless of how a cut is defined, it is always possible to reduce it to a *cutting curve* on the surface, i.e., the line along which a cut would split the surface. Therefore, for every potential cut, the complexity values are integrated along the corresponding cutting curve, and the cut with the highest overall complexity $CPX(\cdot)$ is chosen. The surface is finally split along the chosen cut, and the resulting pieces are inserted in the subdivision hierarchy (a binary tree) as children of the initial surface. The user can explore the tree and possibly request a new cut, executing again the whole procedure over all the leaves of the tree.

This is a general scheme to design effective splitting approaches, every step of the process can be customized according to the kind of surface of interest and to the desired results. In the following, we describe all the operations in detail and explain how we have tuned this framework in order to effectively split stream surfaces.

3.1 The Complexity Measure

The complexity measure $cpx(\cdot)$ is a function that associates a certain complexity value to every $\mathbf{p} \in P$. The meaning of this value depends on how the function is computed. Since our goal is to reduce occlusion, we define the complexity so that $cpx(\mathbf{p})$ represents how much $\mathbf{p}$ conceals the rest of the surface. However, to accurately evaluate such a measure, all the possible view-

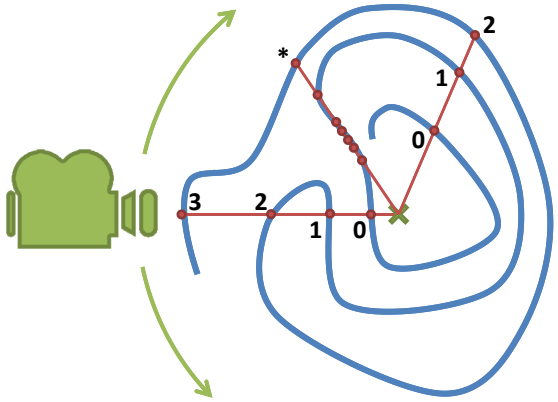


Figure 4: The typical visualization scenario. The camera (in green) moves circularly around the surface (in blue). The complexity measure, shown for a few points, is computed counting the intersections between the surface and the point-to-pivot line segment (in red).

points should be considered, which is too expensive to allow for user interaction. We opted for an approximation based on a simple consideration: datasets are frequently shown using a polar view, with the camera moving circularly around a pivoting point $\mathbf{o}$ placed at the center of the object of interest. Thus, we consider the amount of occlusion generated by $\mathbf{p}$ when the camera is looking directly at it, i.e., when it lies exactly between the camera and the pivot. Let $\mathbf{r} = \mathbf{o} - \mathbf{p}$ be the vector from $\mathbf{p}$ to $\mathbf{o}$, we set

$$cpx(\mathbf{p}) = \|X\| \tag{1}$$

where X is the set of intersection points between $\mathbf{r}$ and the surface mesh.

There is however an issue to solve: if $\mathbf{r}$ is tangent to portions of the surface, $cpx(\mathbf{p})$ can easily degenerate (Figure 4, middle red line). To attenuate this effect, we additionally take into account the angle between $\mathbf{r}$ and the surface normals $nrm(\cdot)$ at the intersection points

$$cpx(\mathbf{p}) = \sum_{\mathbf{x} \in X} \left| \mathbf{nrm}(\mathbf{x}) \cdot \frac{\mathbf{r}}{\|\mathbf{r}\|} \right| \tag{2}$$

Including the importance measure is straightforward. We have to modify the complexity function so that, if the occluded area is highly important, the complexity of the occluding points has to be high as well. We assume that the degree of interest function is a generic attribute $doi(\cdot)$ defined for every $\mathbf{p} \in P$:

$$cpx(\mathbf{p}) = \sum_{\mathbf{x} \in X} doi(\mathbf{x}) \left| \mathbf{nrm}(\mathbf{x}) \cdot \frac{\mathbf{r}}{\|\mathbf{r}\|} \right| \tag{3}$$

For the moment, we assume that $doi(\cdot)$ is defined at the beginning and never changes during the analysis phase; inclusion of interactive brushing techniques will be investigated in the future.

3.2 The Cut Space

The set of potential cuts can be defined in several ways. For example, Karpenko et al. [KLMA10] define it as a set of planes orthogonal to an explosion axis. Li et al. [LACS08], instead, define cuts as the boundaries of the components of the initial object. The fundamental requirement is that the elements of the cut space split the surface in meaningful and easily understandable pieces. In the case of flow data, defining such a space is not trivial: arbitrary cuts with a fixed geometry, such as planes or cubes, can reduce cluttering but the resulting pieces would be of difficult interpretation. Moreover, integral surfaces are not aggregate objects, so their building blocks cannot be easily defined.

One of the main characteristics of stream surfaces is that they have a semantically meaningful parametrization: every point on the surface lies in fact on the trajectory of one of the advected particles. Therefore, every point $\mathbf{p}$ can be associated with two parameters

- the *seeding point* $s(\mathbf{p})$: the location where the related particle has been seeded, expressed as a percentage of the length of the seeding line
- the *integration time* $t(\mathbf{p})$: the time needed by the related particle to travel from the seeding point to $\mathbf{p}$.

The isocurves of these two attributes are actually streamlines and time lines respectively. When a stream surface is split along one of these curves, the resulting pieces are stream surfaces as well. Therefore we define the cut space as the set of streamlines and time lines, corresponding to regular samples of their value ranges.

Notice that $s(\cdot)$ and $t(\cdot)$ are bijections. Therefore, in parameter coordinates, the surface is simply a portion of the $2D$ space, and the cuts become straight line segments parallel to the axis (Figure 5, left).

To improve the versatility of our system, we also provide the possibility of considering isocurves of arbitrary parameters. An example is shown in Figure 5, right, where the integration time has been replaced by the integration length, i.e., the arc length of the trajectory.

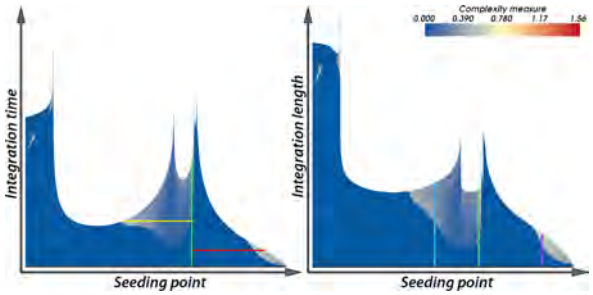


Figure 5: A stream surface from the ABC flow shown in parameter space, with three cuts. (left) parametrization given by the seeding point and the integration time. (right) The integration distance is used instead of the integration time.

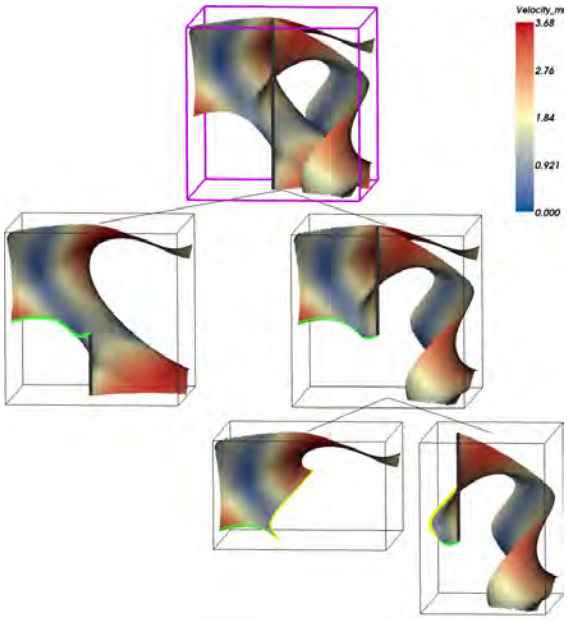


Figure 6: The tree obtained cutting two times a stream surface from the ABC flow. The first cut is made along a streamline (in green) and the second one along a time line (in yellow).

3.3 Surface Cutting

Given the space of potential cuts, we have to determine which cut would result in the most effective reduction of structural occlusion. Recall that the complexity measure has been already evaluated for every point on the surface. Then, we define the overall complexity $CPX(\cdot)$ of a cut Ω as the average complexity along it:

$$CPX(\Omega) = \frac{1}{length(\Omega)} \int_{\mathbf{x}\in\Omega} cpx(\mathbf{x}) \qquad (4)$$

An approximation of this integral is computed in the $2D$ parameter space as explained in Section 5.

The final step consists in selecting the cut with the highest overall complexity and using it to split the surface. However, the proposed complexity measure does not take into account the size of the resulting pieces. Usually, removing a relatively small piece from a large surface does not lead to a significant occlusion reduction. Therefore, we bias the cut selection in two ways: firstly we discard cuts that are shorter than a specified threshold. Then we adjust the complexity of the cuts according to the area ratio of the resulting pieces.

After the optimal cut is selected, the stream surface is split and the resulting pieces are inserted in the subdivision hierarchy as children of the split surface. We never had to modify the mesh structure to get well defined cuts, but, for low resolution models, a triangle splitting procedure may be required.

Notice that, if the surface has already been subdivided, the cut evaluation is performed on all the current pieces. Then, only the piece with the highest complexity cut is split.

The subdivision hierarchy is presented to the user as in Figure 6. At every node of the tree, the corresponding surface piece is displayed. The user can interact with this view to get an overall idea of the generated cuts. Then a single piece can be selected and visualized in a separate view in a focus+context manner: the piece of interest is rendered completely opaque while the rest of the surface can be optionally shown with variable transparency, as in Figure 7, bottom row.

4 DEMONSTRATION

In order to show the capabilities of our visualization system, we used it to explore stream surfaces extracted from one synthetic and two CFD datasets. In the following, we give details about the considered datasets and discuss the most relevant results.

4.1 ABC flow

The *ABC flow* is a synthetic dataset well known in flow visualization [DFH+86]. It is defined as a vector field over the domain $[0,2\pi]^3 \in \mathbb{R}^3$ and the velocities are given by:

$$\mathbf{v}(x,y,z) = \begin{pmatrix} A\sin(z)+B\cos(y) \\ B\sin(x)+C\cos(z) \\ C\sin(y)+A\cos(x) \end{pmatrix} \qquad (5)$$

which are solutions of the Euler equation for inviscid flow. We set $A = sqrt(3)$, $B = sqrt(2)$, and $C = 1$. An

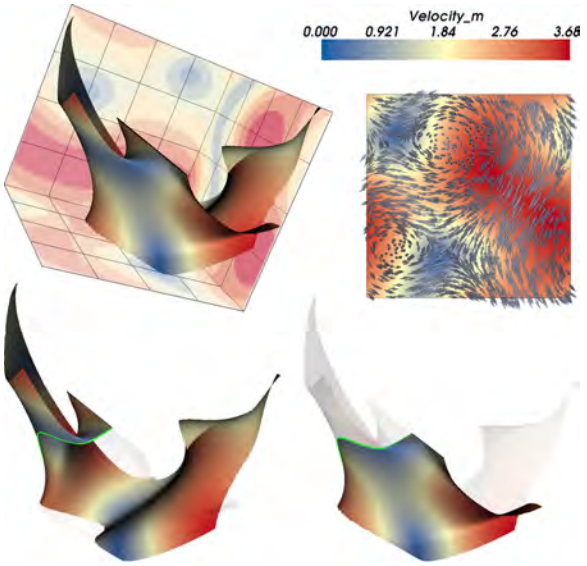


Figure 7: (top left) Overview of the ABC flow dataset, with a stream surface we extracted. (top right) A slice from the ABC flow where the velocity is depicted with glyphs. (bottom) The two pieces obtained by cutting the surface once, using the magnitude of the velocity as DoI. The complementary pieces of surface are shown semi-transparent to provide the context.

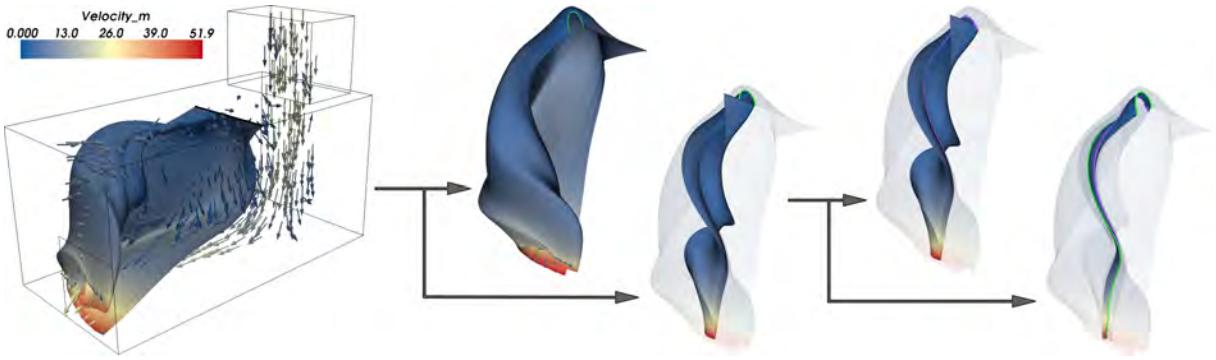


Figure 8: CFD simulation of a fluid flow in a box. The leftmost picture gives an overview of the dataset with the extracted stream surface. The other pictures show the surface after the first and the second cut.

overview of the dataset is given in Figure 7: the top left picture shows the boundaries of the domain and one expressive stream surface we extracted; the top right picture depicts the flow behaviour on the $z = \pi$ plane.

The stream surface under consideration has two almost overlapping areas in the bottom part, one on the left and one on the right. If we do not take into account any DoI, we expect that the splitting procedure separates these areas of the surface. That is exactly what happens after the first cut in Figure 6. The situation is even more interesting if we set the DoI proportional to the velocity magnitude: as can be seen in Figure 7, bottom row, the first cut is made so that the high velocity areas at the bottom right are clearly visible.

4.2 Flow in a box

The second dataset we investigated using our framework is a CFD simulation of fluid flow in a box-like structure. As illustrated in Figure 8, left, the inlet is placed on the far upper side, while the outlet is situated on the front plane, adjacent to both the right and the bottom wall. Vortices and eddies are expected close to where the inlet connects to the box, so we seeded a stream surface in that area.

The surface adequately conveys the rotational behaviour, but, due to self occlusion, it is very difficult to understand what is actually happening in the inner part. After applying a first cut, the more stable piece of the surface is separated from the swirling one, effectively showing the inner vortex (Figure 8, second and third pictures from the left). Requesting an additional cut, the twisting piece is split again (Figure 8, fourth and fifth pictures). This exposes the inner part of the surface and let us analyze the swirling behaviour close to the core of the vortex. Achieving the same goals with traditional techniques, such as transparency or clipping, would have been substantially more difficult.

4.3 Gas leak simulation

The last dataset is a CFD simulation of a gas leak in a closed room on an oil platform. An overview of the architectural structure is given in Figure 2, top. The left and right walls are semi-permeable and, in normal condition, there is an almost constant flow of air in the room, from right to left. After the gas begins leaking, it mixes with air and affects the regular air flow.

The gas/air mixture is described by the *equivalence ratio* (ER), which roughly represents the ratio between fuel and oxidant. In our scenario, where ER is between 0.5 and 1.7 the mixture is flammable, while ER greater than 1.7 means that the mixture cannot burn but it is not breathable either. One of the aspects of interest in this dataset is identifying the locations where there is mixing between air and gas.

We seeded a stream surface in front of the gas leak and observed its behaviour. Two vortices can be easily identified in the top part of the spatial domain and, given their proximity to the leak, they may have a strong influence on the mixing process. Our splitting approach, already at the first cut, correctly separates the branch with the two vortices from the rest of the surface (Figure 2). Figure 9 shows the effect of subsequent cuts: the swirling areas of the surface are effectively

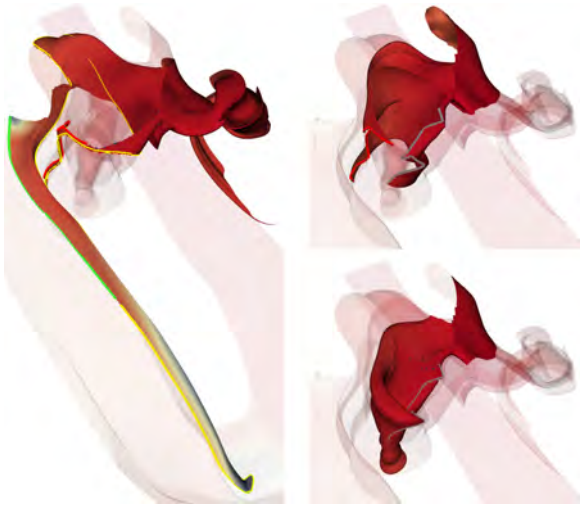


Figure 9: Pieces of stream surface extracted from the gas leak dataset. Iteratively cutting the surface with the proposed approach allows for an easy investigation of the inner areas of the vortices.

Dataset	Vertices	Triangles	Complexity Measure	Best Cut Search	Splitting
ABC flow	42 050	82 329	0.379 s	0.278 s	0.094 s
Box	166 499	322 931	1.466 s	0.582 s	0.362 s
Gas leak	151 320	286 874	1.438 s	0.475 s	0.301 s

Table 1: Summary of the execution time of every step of the pipeline.

subdivided, and the resulting pieces can be more easily investigated and analyzed.

We received positive feedback from a domain expert. Our splitting scheme is deemed effective in simplifying stream surfaces, easing the analysis phase. The approach is considered well suited for the validation of dispersion models and, in general, for the study of turbulence and small scale phenomena.

5 IMPLEMENTATION

The splitting algorithm can be briefly summarized as follows: when a cut is requested, for every current piece of the surface the complexity is computed, the cut space is generated, the best cut is identified and finally the corresponding piece is split. Notice that for every piece, the complexity, the cut space and the best cut can be stored and reused when another cut is requested. In order to maximize the efficiency of our system, the current implementation precomputes all these values for the existing pieces. Therefore, when a cut is requested, the previously computed best cut is used to split the corresponding piece of surface, then the two resulting pieces are analyzed and the next best cut is determined.

If the mesh used to represent the stream surface has a large number of vertices and triangles, determining the best cut can take a considerable time. We aim at supporting user interaction on, at least, surfaces of average size, thus, we introduced various optimizations. First of all, the computation of the complexity measure is based on a ray casting process in the three-dimensional space. This is known to be a highly expensive procedure. But we can exploit the fact that the rays we trace are always directed towards the pivot. We then compute the spherical coordinates (r, ϕ, θ) of every vertex with respect to the pivot: in the resulting spherical space, all the rays we need to trace are parallel to the r axis, which means we have one less dimension to take into account. Moreover, in this space we can use a simple quad-tree to speed up the process.

A similar idea is adopted to approximate the integration of complexity along the cuts. In the $2D$ parameter space, the surface is a flat plane and the cuts are straight lines parallel to the axis (see Section 3.2). Therefore we compute the parameter coordinates of the points and rasterize the transformed surface on a $n \times n$ grid. The parameter n is user specified and determines the size of the cut space. Every row and every column of the resulting image represents a possible cut: evaluating their overall complexity is now a simple image processing procedure.

The time needed to complete any of the steps of the pipeline is heavily dependent on the number of points and triangles of the mesh. This implies that, with the current implementation, the initial surface is the one that requires the most computational efforts to be analyzed. Table 1 summarizes the execution times of every step of the pipeline on the initial surface on a 2.8 GHz CPU. It is clear that the computation of the complexity measure is still the most expensive step despite the optimization. As a matter of fact, the complexity of a vertex is completely independent from the complexity of other vertices, so its computation can be easily performed on the GPU. This will be part of future developments.

6 CONCLUSION AND FUTURE WORK

We propose a novel illustrative flow visualization algorithm which can iteratively split an integral surface while preserving its semantic meaning. The subdivision effectively reduces the structural occlusion caused by the wrapping and twisting of the surface. The resulting pieces are presented in a focus+context fashion, and the relationships between different parts of the surface are conveyed through a subdivision hierarchy. We have applied our visualization system to study one synthetic dataset and two CFD simulations, obtaining meaningful results and receiving positive feedback from a domain expert.

We have already planned a series of changes which will improve different components of our framework. As mentioned in the previous section, we plan to rework the implementation, introducing additional optimizations and executing the parallelizable operations on the GPU. Regarding the visualization, many ideas are being evaluated: e.g., the subdivision tree can be modified in order to present both the hierarchical and the adjacency information between the surface pieces. Moreover, in the focus+context view, it can be useful to show a set of selected pieces instead of just one.

In this paper we have demonstrated our approach applied to stream surfaces, but its extension to path surfaces is straightforward. We believe that the general idea can be applied to many different kinds of surfaces once a suitable cut space has been determined.

ACKNOWLEDGEMENTS

Many thanks the anonymous reviewers for their feedback. We are grateful to Maik Schulze and Holger Theisel for providing the code for the generation of stream surfaces. Special thanks to Josué Quilliou and

GexCon AS for providing the gas leak dataset. Thanks also to AVL for provinding the dataset of the flow in a box. This report has been worked out within the scope of the SemSeg project and we acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number 226042.

REFERENCES

[AS82] R. Abraham and C. D. Shaw. *Dynamics–the geometry of behavior*. Aerial Press, 1982.

[BCP+12] A. Brambilla, R. Carnecky, R. Peikert, I. Viola, and H. Hauser. Illustrative flow visualization: State of the art, trends and challenges. In *Eurographics 2012 State-of-the-Art Reports*, pages 75–94, 2012.

[BG05] S. Bruckner and M. E. Gröller. Volumeshop: an interactive system for direct volume illustration. In *IEEE Visualization 2005*, pages 671–678, 2005.

[BG06] S. Bruckner and M. E. Gröller. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 2006.

[BM10] S. Bruckner and T. Möller. Isosurface similarity maps. *Computer Graphics Forum*, 29(3):773–782, 2010.

[BPS97] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *IEEE Visualization '97*, pages 167–ff., 1997.

[BVG10] J.-P. Balabanian, I. Viola, and M. E. Gröller. Interactive illustrative visualization of hierarchical volume data. In *Proceedings of Graphics Interface 2010*, pages 137–144, 2010.

[BWF+10] S. Born, A. Wiebel, J. Friedrich, G. Scheuermann, and D. Bartz. Illustrative stream surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16:1329–1338, 2010.

[CSC07] C. D. Correa, D. Silver, and Min Chen. Illustrative deformation for data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1320–1327, 2007.

[DFH+86] T. Dombre, U. Frisch, M. Henon, J. M. Greene, and A. M. Soward. Chaotic streamlines in the abc flows. *Journal of Fluid Mechanics*, 167:353–391, 1986.

[dV11] L. da Vinci. The babe in the womb, c. 1511.

[FTAT00] I. Fujishiro, Y. Takeshima, T. Azuma, and S. Takahashi. Volume data mining using 3d field topology analysis. *IEEE Computer Graphics and Applications*, 20(5):46–51, 2000.

[HGH+10] M. Hummel, C. Garth, B. Hamann, H. Hagen, and K. I. Joy. Iris: Illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16:1319–1328, 2010.

[KLMA10] O. Karpenko, W. Li, N. Mitra, and M. Agrawala. Exploded view diagrams of mathematical surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1311–1318, 2010.

[LACS08] W. Li, M. Agrawala, B. Curless, and D. Salesin. Automated generation of interactive 3d exploded view diagrams. pages 101:1–101:7, 2008.

[LMGP97] H. Löffelmann, L. Mroz, M. E. Gröller, and W. Purgathofer. Stream arrows: Enhancing the use of streamsurfaces for the visualization of dynamical systems. *The Visual Computer*, 13:359–369, 1997.

[MLP+10] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[PVH+02] F. H. Post, B. Vrolijk, H. Hauser, R. Laramee, and H. Doleisch. Feature extraction and visualization of flow fields. In *Eurographics 2002 State-of-the-Art Reports*, pages 69–100, 2002.

[RBGV08] P. Rautek, S. Bruckner, M. E. Gröller, and I. Viola. Illustrative visualization: new technology or useless tautology? *ACM SIGGRAPH Computer Graphics Quarterly*, 42:4:1–4:8, 2008.

[RVB+08] M. Ruiz, I. Viola, I. Boada, S. Bruckner, M. Feixas, and M. Sbert. Similarity-based exploded views. In *Smart Graphics*, volume 5166, pages 154–165. 2008.

[VKG05] I. Viola, A. Kanitsar, and M. E. Gröller. Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):408–418, 2005.

[WS05] A. Wiebel and G. Scheuermann. Eyelet particle tracing - steady visualization of unsteady flow. In *IEEE Visualization 2005*, pages 607–614, 2005.

Morphometric Analysis of Mesh Asymmetry

Václav Krajíček[1,2] Ján Dupej[1] Jana Velemínská[2] Josef Pelikán[1]
vajicek@cgg.mff.cuni.cz jdupej@cgg.mff.cuni.cz velemins@natur.cuni.cz pepca@cgg.mff.cuni.cz

[1] Department of Software and Computer Science Education, Charles University in Prague, Malostranské náměstí 25, 11800, Prague, Czech Republic

[2] Department of Anthropology and Human Genetics, Faculty of Sciences, Charles University in Prague, Viničná 7, 12844, Prague, Czech Republic

ABSTRACT

New techniques of capturing shape geometry for the purpose of studying asymmetry in biological objects bring the need to develop new methods of analyzing such data. In this paper we propose a method of mesh asymmetry analysis and decomposition intended for use in geometric morphometry. In geometric morphometry the individual bilateral asymmetry is captured by aligning a specimen with its mirror image and analyzing the difference. This involves the construction of a dense correspondence mapping between the meshes. We tested our algorithm on real data consisting of a sample of 102 human faces as well as on artificially altered meshes to successfully prove its validity. The resulting algorithm is an important methodological improvement which has a potential to be widely used in a wide variety of morphological studies.

Keywords

mesh asymmetry, geometric morphometry, mesh correspondence

1. INTRODUCTION

The geometry of an object reflects many facts about its creation development and purpose. A significant property observed in many biological objects is symmetry. Specifically, bilateral symmetry can be defined as the existence of a plane that splits an object into two identical parts with respect to reflection. This kind of symmetry is exhibited by most living natural objects (see Figure 1). Deviation from bilateral symmetry, asymmetry, can result from various stresses in population or individual development. Evaluation of asymmetry in the human face is useful in various scientific fields like anthropology, plastic surgery, forensic medicine, orthodintics, surgery, anatomy and others. Notably, the quantitative analysis of asymmetry provides important information for treatment planning; e.g. it can be used to determine the target area or the surgical method to be applied [Dam11].

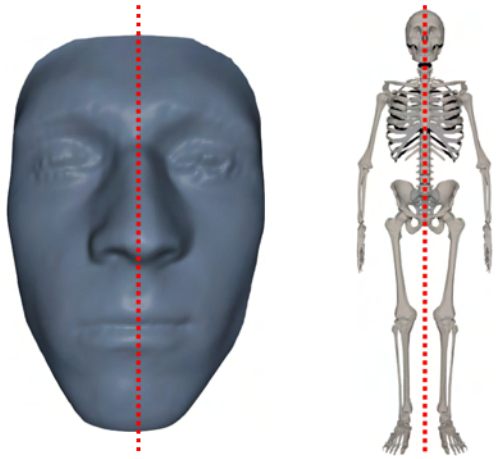


Figure 1: Examples of bilateral symmetry. It can be easily found in many higher and lower species.

Traditional and geometric morphometry, the tool of many fields of natural science such as botany, zoology or anthropology, offers ways to analyze the asymmetry of simple morphometric data such as lengths and landmark locations. However, with the advent of new measuring techniques such as surface and CT scanners to these fields new methods are required to analyze the full measure of information provided by these new data modalities.

Particularly we aim to develop an algorithm to analyze the asymmetry in triangular meshes in a way that is needed for geometric morphometry [Boo97],

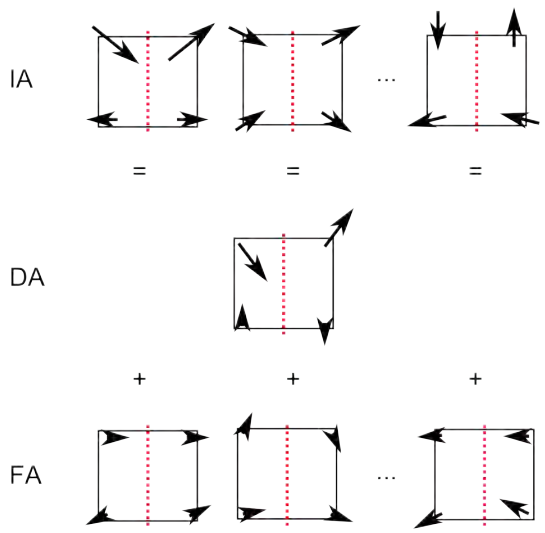


Figure 2: Decomposition of individual asymmetry (IA) into directional (DA) and fluctuating (FA) asymmetry components.

which includes its decomposition into directional and fluctuating asymmetry [Val62] [Pal94] [Kli02]. Before we analyze the group tendency to asymmetry we have to formulate how to interpret individual asymmetry.

Individual asymmetry is the quantified difference in a particular feature from its paired counterpart. Directional asymmetry is then the average of these differences across the studied sample. The statistical significance of this average can be evaluated using standard statistical tools such as the t-test. In other words, directional asymmetry describes the tendency to a certain deviation from the symmetry of the whole group or population. On the other hand, fluctuating asymmetry is defined as the difference between individual asymmetry and directional asymmetry and thus represents the random presence of asymmetry in the individual. Fluctuating asymmetry traits are normally distributed around the mid-sagittal plane. The overall magnitude of the fluctuating asymmetry is generally more significant than its spatial distribution. The process of such decomposition is visualized in the Figure 2.

In geometric morphometry, asymmetry is evaluated on paired features, i.e. paired landmarks and distances. The correspondence of particular features in traditional and geometric morphometry across the sample is known by definition from the homology of the features. Such correspondence is, however, not implicitly defined on triangular mesh data.

In the following chapter, some existing work related to mesh asymmetry analysis and extraction will be introduced. In Chapter 3 we will present the basis of our approach, namely the dense correspondence algorithm introduced by Hutton [Hut01]. Next, in Chapter 4 we thoroughly explain our procedure. We then we demonstrate its results in three scenarios in Chapter 5. Finally, Chapter 6 concludes our work and discusses future extensions.

2. RELATED WORK

Symmetry and asymmetry is an important feature of the human brain which was intensively studied in the past. In the modern era there have been many attempts [Fou11] to analyze MRI images and interpret brain asymmetry with respect to its connection to illnesses, functional abilities or genetics.

There have not been many attempts at automatic analysis of asymmetries in mesh data. One particular approach [Liu03] maps objects of interest onto a surrounding cylindrical surface. In the reference cylindrical coordinate system, the corresponding symmetric points are found with the help of manually placed landmarks. Asymmetry is then deduced from these pairs. This approach obviously works only for simple shapes that can be unambiguously projected onto a cylinder.

A different method [Ola07] assumes the existence of an ideally symmetric template and then maps each subject in the study onto this template using B-spline based non-rigid registration. Construction of the ideal or any symmetric template for a certain group is not trivial. Constructing that template requires the so called mid-sagittal plane about which the template is bilaterally symmetrical.

Another approach [Com08b] constructs a mid-sagittal plane using a modified EM algorithm [Com08a] and uses it to mirror the studied shapes. Asymmetry is then represented as the distance between the corresponding points on the original and mirror shape. Correspondence is found using non-rigid registration, bending the mirror shape to the original.

In geometric morphometry [Boo97], asymmetry was analyzed on landmark datasets [Sch06] by mirroring the landmark configuration and reordering the landmarks so that the mirror and the original can be realigned. The asymmetry is then defined as the difference of an ideally superimposed mirror and the original. More importantly, the approach decomposes the asymmetry in the traditional way studied in biological sciences [Val62] [Pal94].

We used some of these ideas in our proposed method.

3. DENSE CORRESPONDENCE

The individual asymmetry can be computed from the knowledge of matching counterpart features in a mirrored mesh. To calculate the directional and

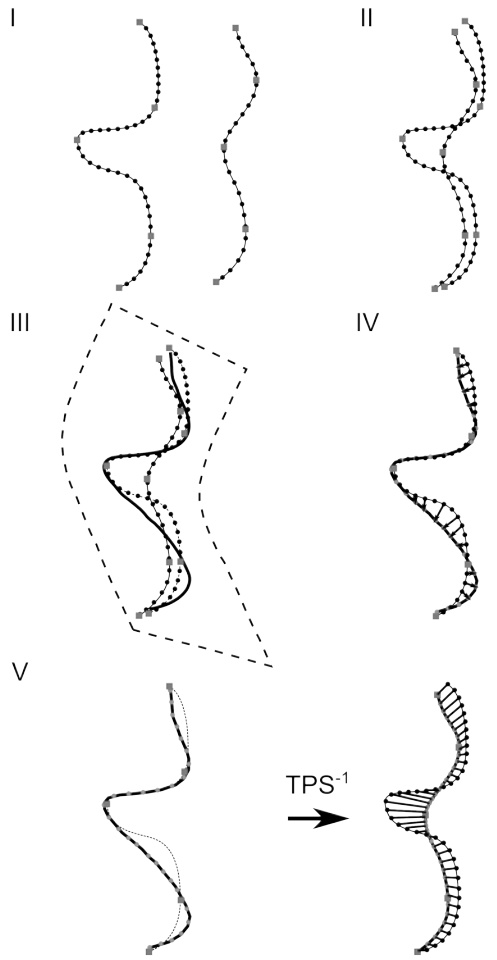


Figure 3: The steps of dense mesh correspondence construction in 2D. I-II) rigid alignment of input data, III) TPS deformation of the moving mesh, IV) nearest neighbor correspondence search, V) inverse TPS deformation of found points

fluctuating asymmetry, the matching of corresponding features in the group must be known.

Finding these matches equates to the construction of correspondence mapping either between a mesh and its mirror image of any two meshes in the group. Generally, any non-rigid mesh registration algorithm could be used for this task.

In the case of biological data we are also able to exploit homology – a unique one-to-one correspondence of certain features of interest. We therefore employ the following mesh correspondence construction.

The dense correspondence construction algorithm by [Hut01] [Hut03] uses sparsely landmarked surfaces. The more landmarks are used, the better the results. If possible, landmarks should be spread evenly over the area of interest. Placing all landmarks in one plane should be avoided as it will reduce the quality of correspondence. This is due to the fact that the spatial deformation describing the correspondences would

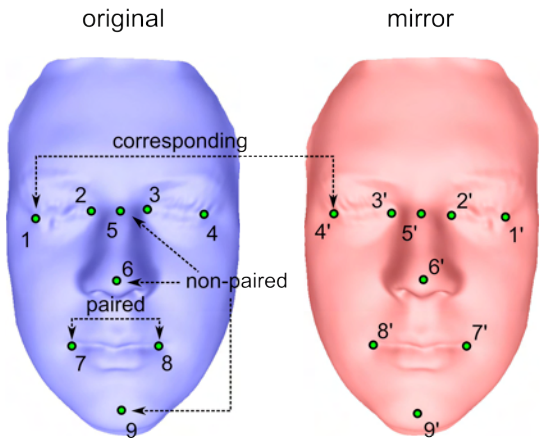


Figure 4: Original and mirror mesh.

not be defined well outside this plane. The mesh against which the correspondences are sought is designated the reference mesh, while the others are referred to as the moving meshes.

First, we rigidly align the moving mesh to the reference mesh by minimizing the squared distance of their respective landmarks using ordinary Procrustes superimposition, a method of rigid registration [Boo97] while preserving unit centroid size.

Second, the aligned moving mesh is bent to fit the reference mesh using thin-plate splines (TPS) interpolation

$$TPS(\vec{x}) = \vec{a}_0 + \vec{a}_1 x + \vec{a}_2 y + \vec{a}_3 z + \sum_{i=0}^{n} \vec{w}_i \varphi(\|\vec{x} - \vec{p}_i\|)$$

where $\vec{p}_i$ are landmark locations on the reference and φ is the radial basis function $\varphi(r) = r^3$. The TPS is fitted to the superimposed landmark pairs from the previous step. This way the meshes are brought as close as possible to each other so that the correspondences can be constructed.

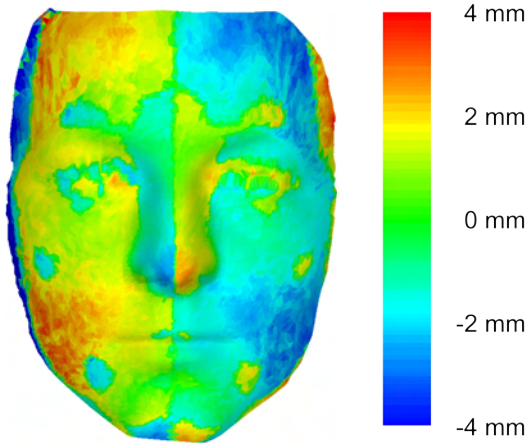


Figure 5: Symmetric face and individual asymmetry captured by the algorithm

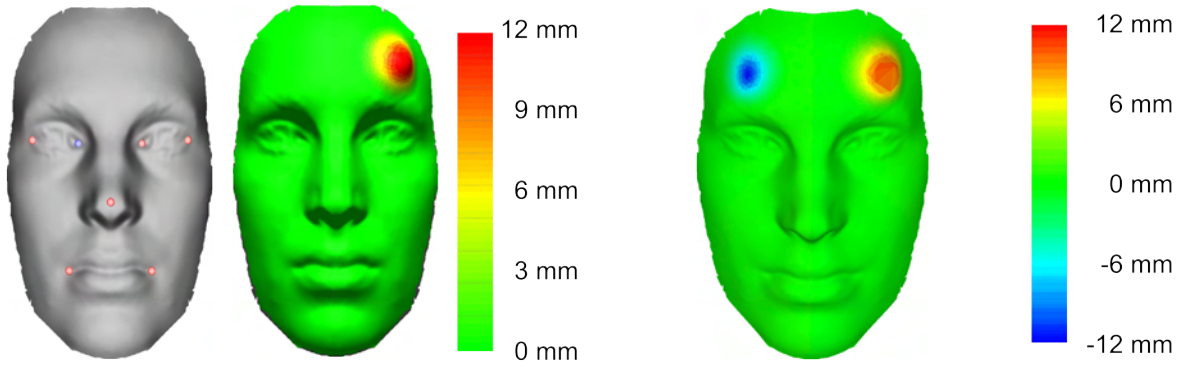


Figure 6: An ideally symmetric face and the artificially added asymmetry (left). Individual asymmetry of artificial test subject detected by the algorithm (right).

In the last step the reference mesh is used to construct a new mesh with the same topology as the reference but the shape of the moving mesh, which is referred to as the correspondence mesh. This is done by finding the closest point (not necessarily vertex) in the reference mesh to each vertex of the moving mesh. The search efficiency can be enhanced with acceleration structures. A kd-tree on all mesh triangles has proven very effective. This process yields the correspondence between the reference mesh's points and the deformed moving mesh. We now need to transform these vertex locations to the space of the original moving mesh. Because the deformed mesh was created with a TPS the original is obtained by inverting that TPS. Since TPS has no analytically defined inversion it must be computed numerically as the solution of the following minimization problem.

$$\underset{\vec{x}\in R^3}{\boldsymbol{arg\ min}} \|TPS(\vec{x})-\vec{y}\|^2$$

This could be solved by any optimization procedure. Because in this case the second derivatives can be easily analytically expressed, the Newton's method is suitable for the problem.

An alternative to the numeric TPS inversion computation is an approximation using barycentric coordinates. The correspondence points in the deformed moving mesh are expressed in barycentric coordinates inside their respective triangles. The same barycentric coordinates are then used to compute the point in the corresponding face of the original moving mesh. A scheme of the correspondence construction procedure is described in Figure 3.

Sometimes the corresponding point is located too far from the vertex in the reference. Then it is not likely that it is a proper correspondence. We can define a threshold distance beyond which the correspondence is not accepted. In that case we simply omit this vertex from the mesh with identical topology.

In order to analyze a group of meshes the correspondences across the whole sample must be constructed. One mesh from the sample is chosen as the reference and the remaining meshes are used as moving meshes to construct correspondences to the reference.

The choice of the reference mesh may have a significant influence on the result. It should be chosen so that it is not too different from the rest of the group. Ideally, it lies in the middle of the group, hence the other moving shapes need to be bent less to align with the reference. The authors of the original algorithm [Hut01] claim that if the input data is random, choosing the first mesh as the reference for the group is as good as choosing any other.

4. MESH ASYMMETRY ANALYSIS

In our approach to capture a group mesh asymmetry we use the concept of decomposing the asymmetry into directional and fluctuating components. Concurrently, dense triangular meshes allow us to express the asymmetry on a very localized level.

Before we can compare the local asymmetries in all meshes, we need to have them transformed into a common coordinate space. This is done by applying a group-wise rigid landmark-based registration, specifically generalized Procrustes superimposition has been used. The meshes' vertices are transformed the same way as the landmarks in the Procrustes superimposition.

The next step is to recompute the meshes to the same number of vertices and the same topology. We used the dense correspondence construction algorithm from the previous section.

Now, the individual mesh asymmetry is computed. The result is the list of directions for every vertex. If every vertex were moved by their respective displacement the mesh would become ideally symmetric. The mirror mesh must be constructed by

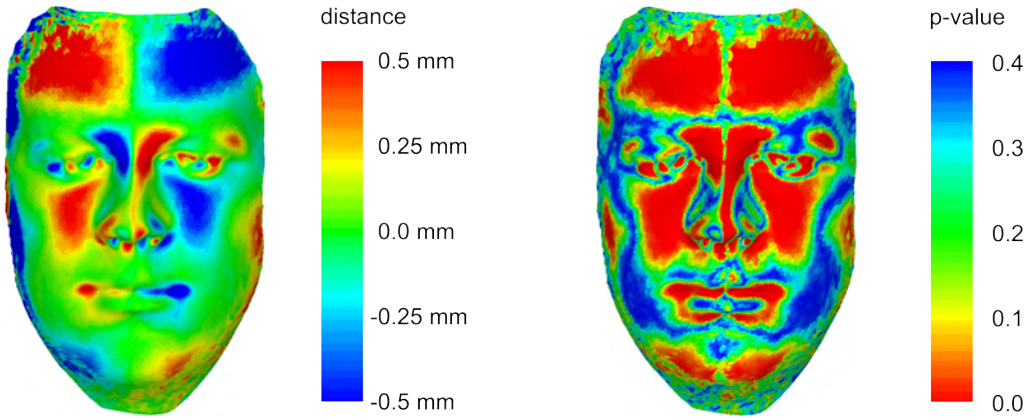


Figure 7: Directional asymmetry (left) and significance map of the asymmetry (right) on 102 human faces.

negating one of the vertex coordinates (see Figure 4). The same is done with landmarks placed on the mesh.

The landmarks are then used to align the mirror mesh back to the original mesh. In order to do that the landmarks must be reordered as they have changed their homologous meaning after the mirroring. e.g. some landmarks on the left side of a bilaterally symmetric mesh became the landmarks on the right side of the mirror mesh. These are the so called paired landmarks; they swap their positions with their mirror counterparts, while the others, the non-paired landmarks are not affected by mirroring.

After mirroring and landmark reordering the mirror meshes are realigned by ordinary Procrustes superimposition and deformed by TPS in order to get mirror meshes closer to the original ones and allow for subsequent correspondence searching. The closest points on the mirror meshes to each vertex of the original correspondence meshes are found using search acceleration structures. Again, we opted for a kd-tree.

The vectors defined by the difference of the original mesh vertices and their closest mirror mesh points is the local measures of asymmetry. Completely symmetrical shapes have identical mirrors and when aligned the distances between the original mesh vertices and the closest points are zero. If asymmetries occur on the mesh the difference between the left and right part of the mesh appears and the distance between a part of the mesh and its mirrored counterpart becomes non-zero. Furthermore, the associated vector holds the information about the direction of the asymmetry, i.e. how the part of the mesh was moved to form the asymmetry. This information is exhibited on either side of a bilaterally symmetrical mesh in the opposite directions. Hence it cannot be said which part of the mesh originated from a symmetric shape and which was altered, if this is the way the analyzed asymmetric shape was created. It can be said that bilateral asymmetry is a symmetric feature. The vector field that represents the displacement of a point on a mesh from where it would lie if the mesh were ideally symmetric is called individual asymmetry.

We visualize the aforementioned vector field on the original mesh with color-coded signed distances (see Figure 5). The sign is the same as that of the dot product of the mesh normal and the vector of individual asymmetry in that point. The color images could be simply interpreted in the following way: red areas lie in the front of the corresponding mirrored counterpart which means that they are larger than the corresponding paired counterpart while blue areas are smaller and lie behind the aligned mirrored counterpart. The areas that are close to green are not significantly larger or smaller. This interpretation does not include any information about the direction

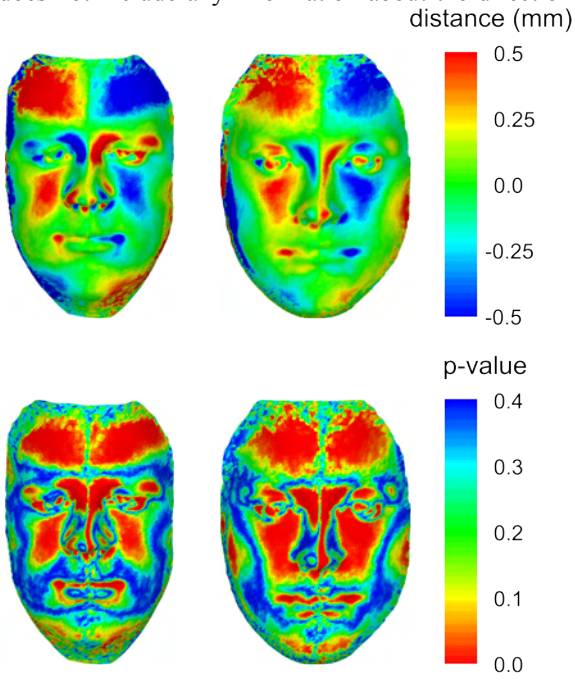


Figure 8: Directional asymmetry of male (left) and female (right) subgroups.

of the asymmetry. This sort of visualization is also known as clearance vector mapping and is useful in quantifying the facial surface asymmetries in the areas where anthropometric landmarks are scarce. The volume of detected asymmetry is potentially significant in patients who may have their unilateral facial deficiencies corrected using injections or implants [OGr99].

All individual asymmetries are already aligned group-wise, therefore, directional asymmetry is computed as the average of all corresponding individual asymmetry vectors. The length of the respective individual asymmetry vectors is the same for all the meshes and they correspond to each other per element as the meshes were reconstructed to have the same topology. The directional asymmetry is visualized the same way as individual asymmetry.

Fluctuating asymmetry is computed as the difference of individual asymmetry and directional asymmetry. Its visualization is also based on color coded distances without the consideration of the sign for direction as was done for individual and directional asymmetry above. As stated in the previous chapter, we are more interested in the overall magnitude of the fluctuating asymmetry which is computed as the sum of squared distances of the fluctuating asymmetry vectors. It can be compared across the group and if normalized by the number of vectors, or between groups just as well.

To prove that the directional asymmetry reflects the global trend of the group, and is not the result of randomness in the group, it has to be tested statistically. A standard t-test is performed on the signed lengths of corresponding individual asymmetry vectors. The significance map can then also be visualized. This way of interpretation is especially important for particular research in biological sciences.

The direction of the individual asymmetry vector is also important property that should be taken into account. In order to do so we define the local orientation difference asymmetry measure which is equal to cosine of angle between corresponding individual asymmetry vectors.

The lengths of individual asymmetry vectors and local orientation difference asymmetry measure can be summarized into total asymmetry and total orientation asymmetry.

$$TA = \frac{1}{N}\sum_{i=0}^{N} \left\| \vec{a}_i \right\|^2$$

$$TOA = \frac{1}{N}\sum_{i=0}^{N} \vec{n}_i \cdot \vec{m}_i$$

5. RESULTS

Our semiautomatic landmark-based approach reflects the needs of scientists from fields like archaeology and anthropology where the homology of certain feature points is advantageous. It is often the case that mesh datasets together with their respective landmark configurations already exist and the results of landmark-only based studies have already been published so some comparison of results can be done. All these facts are the motivation supporting our solution over completely autonomous algorithms, e.g. employing nonrigid mesh registration.

In the first test it will be demonstrated how the proposed algorithm approximates individual asymmetry on an artificial dataset where the ground truth is well known. A symmetric face model was created by cutting a real face in half approximately along its medial axis mirroring one of the halves and stitching it to its original. Then the symmetric face was locally deformed, bulged on the left side of the forehead. Shell distance measured by a software tool, RapidForm XOS in our case, was used as the ground truth (see the left side of Figure 6). In this case, if the deformation does not affect the areas with landmarks, the algorithm can uncover the asymmetry perfectly (see the right side of Figure 6). If the location of a landmark is distorted by a nearby asymmetry the error of alignment by the generalized Procrustes superimposition will be distributed among all

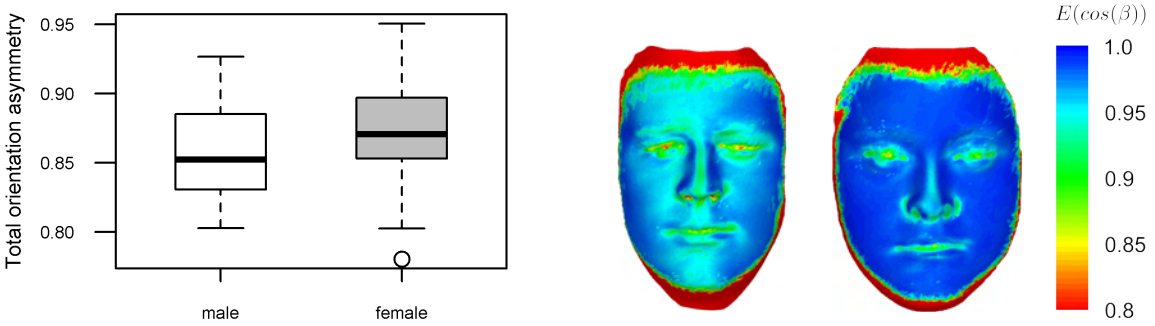


Figure 9: Difference in total orientation asymmetry of male and female subgroups is statistically significant ($p < 0.05$). The orientation the asymmetry is locally unevenly distributed on the male faces while being uniform almost everywhere on female faces.

landmarks. Even in this case the algorithm still reveals the asymmetry relatively well. In case of large asymmetries all across the mesh it would be difficult for any method to recover results close to ground truth as in this kind of problem it is highly ambiguous what the original symmetric shape is. Therefore in practice, landmarks are usually placed on stable locations. These locations may exhibit asymmetry as well, it is however assumed that the user will choose a landmark configuration whose own asymmetry will have the least impact on the results.

In the second test a group of 102 real faces were analyzed. The faces were captured by a *InSpeck 3D MegaCapturor II* scanner with 0.4 mm resolution in the direction of its optical axis. The resulting meshes with approximately 100k-200k triangles were cleaned and trimmed from border areas and finally decimated to approximately 20k-30k triangles. Our landmark configuration contains nine landmarks situated in the corners of mouth, eyes, and on the tip and around the nose. The landmarks were placed by an expert. The resulting directional asymmetry can be seen in the left side of Figure 7. The significance map of the asymmetry is in right side of Figure 7. The group includes both male and female subjects. Each sex can be analyzed separately and compared (see Figure 8). Local asymmetry information can be summarized into a single value per specimen called total asymmetry. Neither individual asymmetry nor total asymmetry discriminates between sexes in our sample. On the other hand, the total orientation asymmetry shows the difference rather clearly (see Figure 9). This finding is confirmed by results from [Liu03] which indicates that the orientation asymmetry is important in comparative studies. Our method is new in ability to capture asymmetry and correspondence in more complex shapes.

In the third (Figure 10) test we used 50 scans of human palate (a convex surface surrounded by dental arc). The shape of this particular structure is studied in order to describe impact of the therapy on patients with cleft of the lip and palate. Specifically the palate has an altered shape and its further development is influenced by inadequate growth of maxilla. The shape of the palate has great individual variation. The above-described methodology is useful for the comparison of the palatal shape depending on the type and extent of the cleft, the utilized surgical method and the surgeon's proficiency, as well as numerous other factors associated with surgical and orthodontic treatment [Sma03]. The shape of the palate is a problematic object from the view of geometric morphometry as the only apparent landmarks can be placed at the locations of teeth.

The whole algorithm is not very computationally demanding. The analysis of all 102 faces took approximately 283 seconds on Intel Core i7 machine. The computation consists of many independent parts that can be computed in parallel. For instance the group correspondence construction in fact involves N independent processes, N being the count of meshes in the group. Similarly, individual mesh asymmetry can be computed independently for each mesh.

6. CONCLUSIONS

The dense correspondence algorithm has been used in many geometric morphometry studies, e.g. [Ham04] [Bej12] [Vel12]. It extends the ability of GMM methodology to capture shapes from simple primitives such as homologous landmark to triangular meshes representing the whole surface of the object. We continued in this trend to study asymmetry of groups of shapes in a novel way but using the traditional approach of decomposition into directional and fluctuating asymmetry.

The most significant contribution of our work is the application of dense correspondence mapping, as introduced in [Hut01], to map asymmetries onto complex geometry, as opposed to [Liu03] that only used cylindrical surfaces. Furthermore, our approach utilizes landmark-based registration, which makes it more adjustable in certain scenarios than most automatic algorithms [Com08b].

From a practical point of view and in comparison to commercially available tools implementing dense correspondence modeling algorithms such as

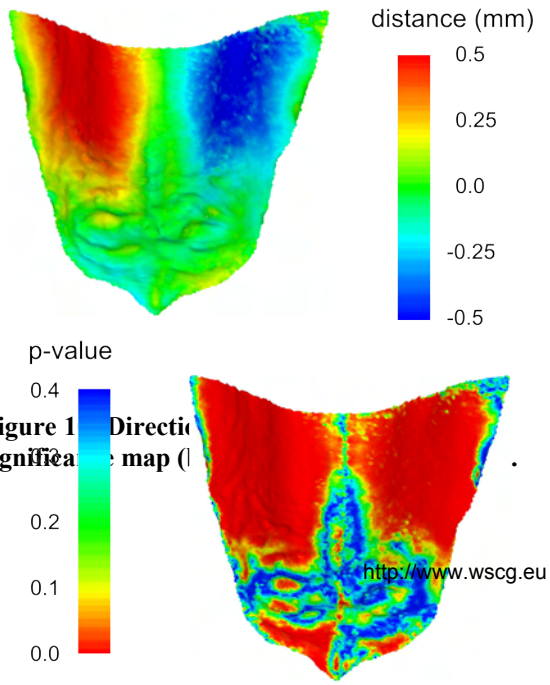


Figure 1 Directi
significa map (

MorphoStudio 3.0 (from BioModeling Solutions, 2006) we sped up the correspondence matching by employing a kd-tree search data structure.

7. ACKNOWLEDGEMENTS

Our thanks goes to the members of Laboratory of 3D Analytical and Visualization methods, Charles University in Prague for providing us with real data. This research has been supported by GAUK 309611 research grants from the Grant Agency of Charles University in Prague and MSM 0021620843 from the Ministry of Education, Youth and Sports of the Czech Republic.

8. REFERENCES

[Bej12] Bejdová, Š., Krajíček, V., Peterka, M., Trefný, P., and Velemínská, J. Variability in Palatal Shape and Size in Patients with Bilateral Complete Cleft Lip and Palate Assessed Using Dense Surface Model Construction and 3D Geometric Morphometrics, J Cranio Maxill Surg, 40 [3]:201–208, 2012.

[Boo97] Bookstein, F.L. Morphometric Tools for Landmark Data : Geometry and Biology, Cambridge University Press, 1997.

[Com08a] Combes, B., Hennessy, R., Waddington, J., Roberts, N., and Prima, S. Automatic Symmetry Plane Estimation of Bilateral Objects in Point Clouds, In IEEE Conference on Computer Vision and Pattern Recognition (CVPR'2008), Anchorage, USA, IEEE, 2008.

[Com08b] Combes, B. and Prima, S. New Algorithms to Map Asymmetries of 3D Surfaces, In Proceedings of the 11th International Conference on Medical Image Computing and Computer-Assisted Intervention - Part I, Berlin, Heidelberg, Springer-Verlag, pp.17–25, 2008.

[Dam11] Damstra, J., Oosterkamp, B.C.M., Jansma, J., and Ren, Y. Combined 3-dimensional and Mirror-image Analysis for the Diagnosis of Asymmetry, Am J Orthod Dentofac, 140 [6]:886–894, 2011.

[Fou11] Fournier, M., Combes, B., Roberts, N., Keller, S., Crow, T.J., Hopkins, W.D., and Prima, S. Surface-based Method to Evaluate Global Brain Shape Asymmetries in Human and Chimpanzee Brains, In Proceedings of the 8th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Chicago, USA, IEEE, pp.310–316, 2011.

[Ham04] Hammond, P., Hutton, T.J., Allanson, J.E., Campbell, L.E., Hennekam, R.C.M., Holden, S., Patton, M.A., et al. 3D Analysis of Facial Morphology, Am J Med Genet, 126A [4]:339–348, 2004.

[Hut01] Hutton, T.J., Buxton, B.F., and Hammond, P. Dense Surface Point Distribution Models of the Human Face, In Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis, Kauai, Hawaii, IEEE, pp.153–160, 2001.

[Hut03] Hutton, T.J., Buxton, B.F., Hammond, P., and Potts, H.W.W. Estimating Average Growth Trajectories in Shape-space Using Kernel Smoothing, IEEE Trans Med Imaging, 22 [6]:747–753, 2003.

[Kli02] Klingenberg, C.P., Barluenga, M., and Meyer, A. Shape Analysis of Symmetric Structures: Quantifying Variation Among Individuals and Asymmetry, Evolution, 56 [10]:1909–1920, 2002.

[Liu03] Liu, Y. and Palmer, J. A Quantified Study of Facial Asymmetry in 3D Faces, In Proceedings of the IEEE International Workshop on Analysis and Modeling of Faces and Gestures, Nice, France, IEEE, pp.222–229, 2003.

[OGr99] O'Grady, K.F. and Antonyshyn, O.M. Facial Asymmetry: Three-Dimensional Analysis Using Laser Surface Scanning, Plast Reconstr Surg, 104 [4]:928–937, 1999.

[Ola07] Ólafsdóttir, H., Lanche, S., Darvann, T.A., Hermann, N.V., Larsen, R., Ersbøll, B.K., Oubel, E., et al. A Point-wise Quantification of Asymmetry Using Deformation Fields: Application to the Study of the Crouzon Mouse Model, In Proceedings of the 10th International Conference on Medical Image Computing and Computer-assisted Intervention, Berlin, Heidelberg, Springer-Verlag, pp.452–459, 2007.

[Pal94] Palmer, A.R. Fluctuating Asymmetry Analyses: A Primer, In Proceedings of the International Conference on Developmental Instability: Its Origins and Evolutionary Implications, Dordrecht, The Netherlands, Kluwer, pp.335–364, 1994.

[Sch06] Schaefer, K., Lauc, T., Mitteroecker, P., Gunz, P., and Bookstein, F.L. Dental Arch Asymmetry in an Isolated Adriatic Community, Am J Phys Anthropol, 129(1):132–42, 2006.

[Sma03] Šmahel, Z., Trefný, P., Formánek, P., Müllerová, Ž., and Peterka, M. Three-Dimensional Morphology of the Palate in Subjects With Isolated Cleft Palate at the Stage of Permanent Dentition, Cleft Palate-Cran J, 40 [6]:577–584, 2003.

[Val62] Van Valen, L. A Study of Fluctuating Asymmetry, Evolution, 16 [2]:125–142, 1962.

[Vel12] Velemínská, J., Bigoni, L., Krajíček, V., Borský, J., Šmahelová, D., Cagáňová, V., and Peterka, M. Surface Facial Modelling and Allometry in Relation to Sexual Dimorphism, Homo, 63 [2]:81–93, 2012.

Preprocessing for Quantitative Statistical Noise Analysis of MDCT Brain Images Reconstructed Using Hybrid Iterative (iDose) Algorithm

Petr Walek
Dept. of Biomedical Engineering
Brno University of Technology
612 00, Brno, Czech republic
walek@feec.vutbr.cz

Jiri Jan
Dept. of Biomedical Engineering
Brno University of Technology
612 00, Brno, Czech republic
jan@feec.vutbr.cz

Petr Ourednicek
Philips Czech Republic
155 00 Prague, Czech Republic
petr.ourednicek@philips.com

Jarmila Skotakova
Children's Hospital - Faculty Hospital
Masaryk University
625 00, Brno, Czech Republic
j.skotakova@post.cz

Igor Jira
Children's Hospital - Faculty Hospital
Masaryk University
625 00, Brno, Czech Republic
igor.jira@volny.cz

ABSTRACT

Radiation dose reduction is a very topical problem in medical X-ray CT imaging and plenty of strategies have been introduced recently. Hybrid iterative reconstruction algorithms are one of them enabling dose reduction up to 70 %. The paper describes data preprocessing and feature extraction from iteratively reconstructed images in order to assess their quality in terms of image noise and compare it with quality of images reconstructed from the same data by the conventional filtered back projection. The preprocessing stage consists in correction of a stair-step artifact and in fast, precise bone and soft tissue segmentation. Noise patterns of differently reconstructed images can therefore be examined separately in these tissue types. In order to remove anatomical structures and to obtain the pure noise, subtraction of images reconstructed by the iterative iDose algorithm from images reconstructed by the filtered back projection is performed. The results of these subtractions called here residual noise images and are the used to further extract parameters of the noise. The noise parameters, which are intended to serve as input data for consequent multidimensional statistical analysis, are the standard deviation and power spectrum of the residual noise. This approach enables evaluation of noise properties in the whole volume of real patient data, in contrast to noise analysis performed in small regions of interest or in images of phantoms.

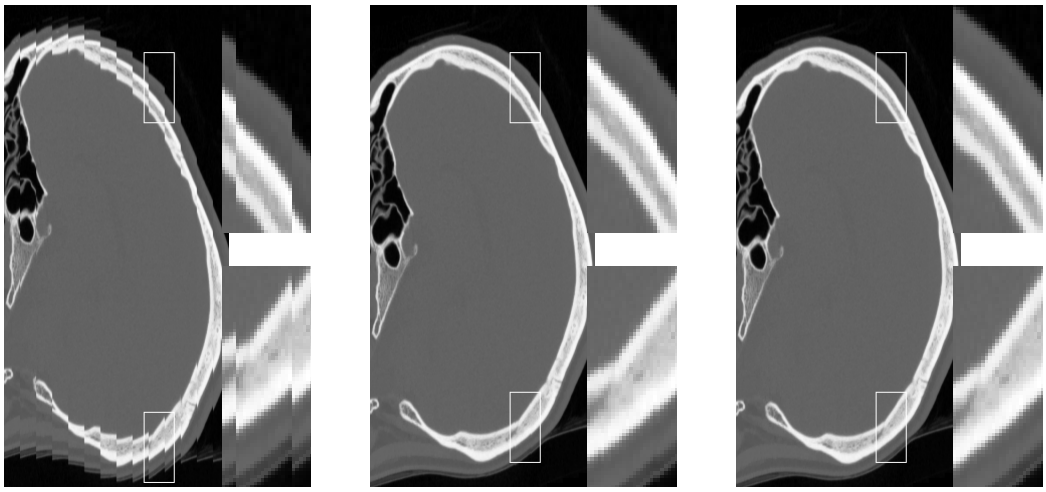
Keywords

X-ray computed tomography, dose reduction, skull segmentation, noise power spectrum.

1 INTRODUCTION

Multidetector X-ray computed tomography (MDCT) imaging is very important for medical diagnostics and quantity of pathological states diagnosed by a MDCT is steadily increasing. This fact causes, in conjunction with rising accessibility of MDCT examinations, increase of the average population radiation exposure which, in spite of unexceptionable diagnostics outcome, constitutes certain health risk especially for pediatrics patients. Effective dose introduced by each MDCT scan depends on many factors and nowadays usually falls within range from 1 to 14 mSv which can be considered as a high value in comparison with the annual dose received from natural sources in the Czech Republic (2.5 mSv).

In order to be compliant with the ALARA principle each of the major MDCT manufacturers have focused their research on as large radiation dose reduction as possible. As a result of this increased effort there have been introduced new strategies for reducing radiation dose, for example tube current modulation (in both angular and longitudinal directions), elimination of over-ranging effect, dual energy scanning, bowtie filtering and replacement of a filtered back projection (FBP) by iterative reconstruction algorithm [MPB+09], [Goo12]. Among a range of mentioned methods the iterative re-



(a) (b) (c)

Figure 1: Saggital slice of brain image (with magnified sections): (a) original slice, (b) slice after registration by phase correlation, (c) slice after registration by gradient descent optimization.

construction one takes exceptional position by producing quality images, even when drastic radiation dose reduction (up to 70%) is applied [FB11]. Such a dose reduction is alowed by inclusion of photon counting statistics and models of acquisition process into reconstruction. So far, each of the available iterative reconstructions are vendor specific and further details about used algorithms are unknown. General description of iterative reconstruction methods can be found in [BKK12] and references therein.

Many studies dealing with problem of quality evaluation of iteratively reconstructed images have been proposed recently. These studies are targeted either to assessment of image quality in small regions of interest in real patient data [MNS+10] or to evaluation of images acquired by scanning of phantoms [MGB+12]. Former approach utilizes information only from spatially limited range and thus can not affect whole complexity of image noise, e.g. differences between noise in anatomical structures. The phantom approach analyzes noise properties in homogeneous regions of artificial images and there is difficulty to relate results obtained by this approach to clinical practice. In order to overcome previously mentioned drawbacks a new way of extraction of noise parameters from whole volume of real patient data is presented in this paper.

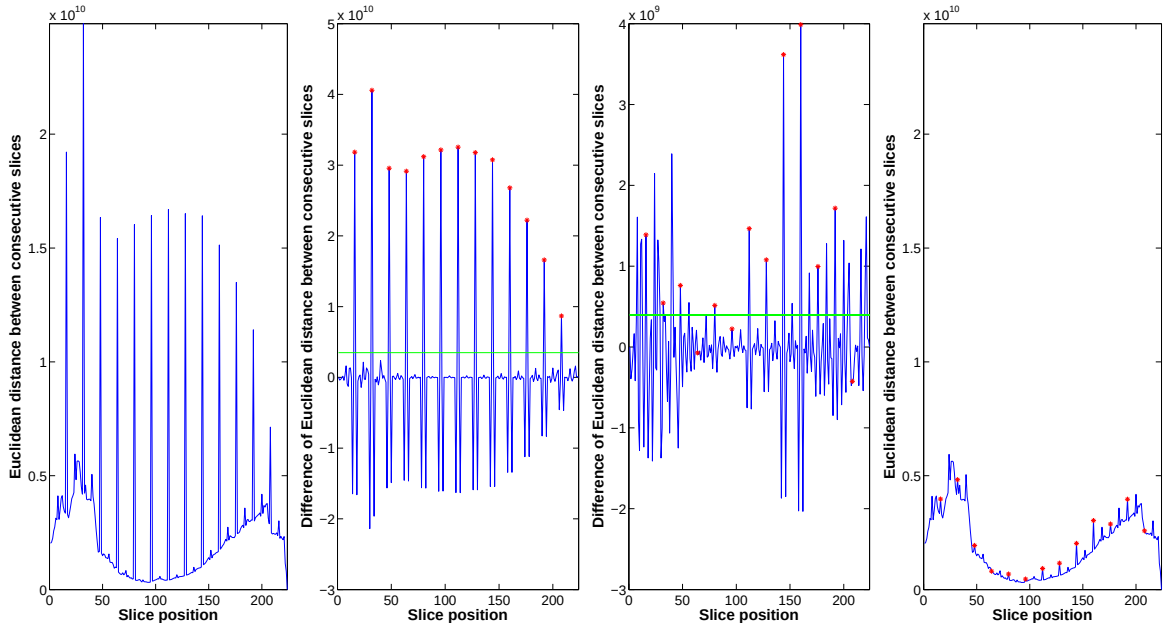
2 DATA ACQUISITION

Our study is targeted to head MDCT images, acquired by the Philips Brilliance CT 64-channel scanner and reconstructed by a prototype of the Philips iterative reconstruction method called iDose during ordinary operation of radiological center in Children's hospital in Brno. Acquired raw data were once reconstructed by the conventional filtered back projection and four times using the iDose algorithm, every time with differently adjusted parameters. The parameters adjusted before each reconstruction are inclusion level of iDose reconstruction expressed in percents (chosen to be 30 %, 50 % and 70 % and in this paper labeled as an ID30, ID50 and ID70), and Multi Resolution which was turned on only together with the iDose level ID70 (in this paper labeled as an ID70MR). Meaning of iDose levels is folowing, images reconstructed by FBP have equal standard deviation of noise as images acquired with 30 % less dose and reconstructed by ID30.

A statistical data set contains forty patients uniformly divided into male and female, aged in range from three months to sixty years. A certain group of patients was scanned with regular dose according to a scanning protocol, other group also with regular dose but in a high quality imaging mode (HQ) and the last group in a high quality mode with radiation dose reduced about 30 % (30HQ). Note that dose reduction was obtained by a uniform reduction of tube current.

3 CORRECTION OF A STAIR-STEP ARTIFACT

Acquired images suffer from the very severe stair-step artifact especially after three-dimensional reformatting to a sagittal plane as can be clearly seen in Fig. 1a. In general a stair-step artifact is caused by using wide collimation and a non-overlapping reconstruction interval especially using multisection scanning [BK04]. Artifact introduces translational shift into sub-volumes located identically according to sections acquired in one gantry rotation during multisection scanning. Such a shift cause many artificial edges in a sagittal plane which are able to harm further noise power spectra analysis by introducing artificial high frequencies.



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

Figure 2: Euclidean distance between consecutive slices (EDCS): (a) original image, (b) difference of EDCS with detected marginal slices (red stars) and threshold (green line), (c) difference of EDCS after registration using phase correlation, (d) EDCS of finally corrected image

Positional detection of translated sub-volumes margins

The first step in correction of the stair-step artifact is positional detection of margins of displaced sub-volumes. Positions of marginal slices are detected by evaluation of the Euclidean distance similarity function (1), computed between consecutive slices. Variables **a** and **b** in this equation means pixel intensities rearranged to a vector and N is a number of pixels in images.

$$C_E(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{i=0}^{N}(a_i - b_i)^2} \tag{1}$$

Resulting vector of Euclidean distances as a function of slice positions can be seen in Fig. 2a. Despite of clearly visible peaks in the similarity function there is also a slow and strong trend which can possibly preclude the detection, and is removed by differentiation of this curve. Resulting difference of the similarity function can be seen in Fig. 2b and a peaks detection algorithm is applied (note that as a peak is labeled each position in the vector with a bigger value than their neighbors). Detected peaks are thresholded, threshold is determined as the mean of absolute values of the vector (depicted as a green line in Fig. 2b), thereby the most significant peaks, representing positions of the most dissimilar consecutive slices, are obtained. The last step is a determination of a patient table translational increment after one rotation of a gantry which corresponds with sizes of mutually translated sub-volumes. Translational incerement of patient table is computed as the median of vector containing distances between the neighboring detected peaks. Finally detected margins of sub-volumes are plotted on Fig. 2b as red stars.

Registration of displaced sub-volumes

Once positions of mutually translated sub-volumes margins are known registration of sub-volumes is performed. Taking into account a character of the stair-step artifact (a simple translation of sub-volumes) phase correlation technique, originally proposed in [KH75], is chosen as a basis for registration. This method is based on a Fourier shift property stating that a planar shift between two functions is expressed in a Fourier domain as a linear phase difference. Let us take two functions $f_1(x,y)$, $f_2(x,y)$ and suppose that they vary only by a translation about Δx and Δy

$$f_2(x,y) = f_1(x-\Delta x, y-\Delta y). \tag{2}$$

Using Fourier shift property equation (2) can be restated to

$$F_2(u,v) = F_1(u,v).e^{(-i(u.\Delta x + v.\Delta y))} \tag{3}$$

where

$$F_i(u,v) = DFT_{2D}(f_i(x,y)). \tag{4}$$

According to equation (3) shifting of image does not influence its amplitude spectrum. Phase correlation can

be calculated as a inverse Fourier transform of a normalized cross power spectrum

$$p(x,y) = DFT_{2D}^{-1}\left[\frac{F_2(u,v).F_1(u,v)^*}{|F_2(u,v).F_1(u,v)|}\right]. \qquad (5)$$

This phase correlation matrix contains a strong impulse in position $[\Delta x, \Delta y]$ which is detected as the strongest peak. Vector of translation parameters $[\Delta x, \Delta y]'$ for each sub-volume is known and alignment can be performed in a very simple manner as $[x,y]' + [\Delta x, \Delta y]'$. Phase correlation, in its basic form, cannot determine sub-pixel shifts and registration therefore cannot be sufficient, see Fig. 1b. After registration of sub-volumes by phase correlation, difference of Euclidean distance between consecutive slices is computed and thresholded again, see Fig. 2c. Euclidean distances between sub-volumes margins (labeled as red stars) above threshold are then minimized by gradient descent optimization method, providing final correction of stair-step artifact, see Fig. 1c and Fig. 2d.

4 SEGMENTATION OF SKULL AND SOFT TISSUE

Very interesting findings may be done if noise properties of differently reconstructed images are examined separately for hyperdense and hypodense structures (i.e. bones and soft tissue). An automatic, reliable and fast segmentation algorithm is therefore needed which should be capable to distinguish between bones and soft tissue even in a complex structure of a basis cranii and segment not only cortical however also trabecular parts of bones. Distinction between soft tissue and bones is carried out in the following manner:

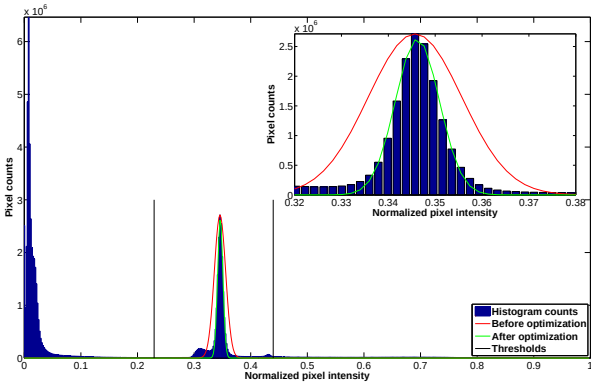


Figure 3: Brightness histogram of the whole brain volume (blue bar graph). Initial Gaussian curve (red plot) is fitted on soft tissue peak (green plot) and final thresholds are depicted as the black lines

- Segmentation of cortical bones.
- Adding trabecular bones parts into segmentation.
- Segmentation of surrounding air and sinuses.
- Segmentation of soft tissue by calculating a complement to segmented bones and surrounding air.

Only the first and the second steps deserves closer attention and are in detail discussed in next two subsections, on the other hand the third step is very similar to the first and the fourth is simple computation of a complement to two binary images (bones and surrounding air segmentations).

Segmentation of cortical bones

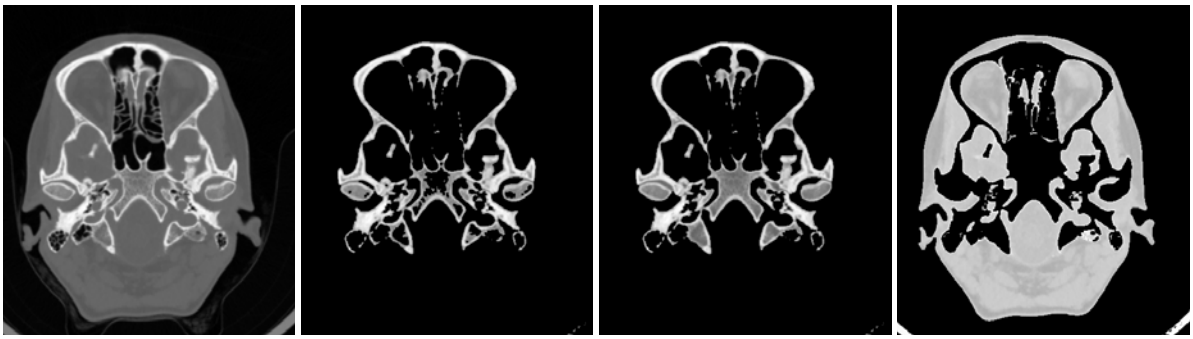
The simplest and fastest method for segmenting cortical bones parts is the intensity thresholding. A threshold is needed for this operation and probably the best way for its automatic determination is evaluation of the image histogram. A typical brightness histogram of the whole brain volume comprises only two distinct peaks, and can be seen in Fig. 3 plotted as a blue bar graph, note that pixel intensities are normalized to be in interval $\langle 0,1 \rangle$. The peak situated at lower intensities belongs to representation of surrounding air and sinuses, while second significant peak belongs to a representation of soft tissue. Intensities belonging to bones are spread over a wide range hence there is no distinct detectable peak. Threshold for cortical bones segmentation is therefore derived from a position of soft tissue peak which is detected in similar way as peaks in chapter 3. Peak with the second highest value is considered to be representation of soft tissue. Detected position of the soft tissue peak serves as a mean μ and magnitude as parameter a of initial Gaussian function (6) used to approximate properties of soft tissue lobe (variance σ is initially selected as 0.01).

$$f(x) = a.e^{\left(-0.5\left(\frac{x-\mu}{\sigma}\right)^2\right)} \qquad (6)$$

The initial Gaussian curve is depicted in Fig. 3 (please notice detailed plot) as a red curve and is optimized by a least-squares curve fitting algorithm in order to find optimal parameters μ and σ (green curve in Fig. 3). Thresholds for segmentation are empirically determined as $\mu - 25.\sigma$ for surrounding air and $\mu + 20.\sigma$ for bones (black lines in Fig. 3). In this way thresholds for bones and surrounding air segmentation are determined automatically and independently on the input data.

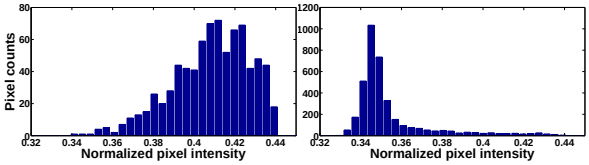
Classification of trabecular bones

Intensities (i.e. Hounsfield units or tissue density) in trabecular parts of bones are partially overlapped with intensities of soft tissue, therefore simple thresholding is only capable to segment cortical parts of bones as can be seen in Fig. 4b (note that resulting binary masks 4b,



(a) Original slice (b) After intensity thresholding (c) Final bones segmentation (d) Soft tissue segmentation

Figure 4: Example of skull and soft tissue segmentation



(a) Histogram of trabecular bone (b) Histogram of Soft tissue

Figure 5: Typical histograms of holes manually classified as soft tissue and trabecular bone

4c and 4d are in this view multiplied with the original slice 4a) and because of that areas being zero fully surrounded by values of one (in this paper called "holes") appears instead trabecular parts of bones. Separation of holes by a boundary tracking technique is therefore next step followed by decision if particular hole represents soft tissue or trabecular bone. As stated before intensities of soft tissue and trabecular bones are partially overlapping, nevertheless their histograms differ in shapes, typical histograms of soft tissue hole and trabecular bone are depicted in Fig. 5. Histograms of trabecular bones parts are, in comparison with soft tissue ones, more compact (histogram counts are smoother) and skewed towards higher intensities. Shape of a particular histogram is objectified by five parameters: entropy (7), compactness (8), relative position of the histogram mean according to position of soft tissue peak in histogram of whole volume (9), skewness (10) and kurtosis (11). In each of the following equations N means sum of all counts in bins (i.e. number of pixels in hole), i is a bin mark and x_i means counts in the bin marked as i.

$$S = -\frac{1}{N}\sum_{i=0}^{n-1} x_i \log(x_i). \tag{7}$$

$$C = \frac{1}{N}\sum_{i=0}^{n-1} \frac{x_i}{\max(x)} \tag{8}$$

$$P_{rel} = \frac{P_{pos}}{\mu}; \quad \mu = \frac{1}{N}\sum_{i=0}^{n-1} x_i i. \tag{9}$$

$$\gamma_1 = \frac{\frac{1}{N}\sum_{i=0}^{n-1}(i-\mu)^3}{\left[\frac{1}{N}\sum_{i=0}^{n-1}(x_i i)^2 - \mu^2\right]^{\frac{3}{2}}}. \tag{10}$$

$$\gamma_2 = \frac{\frac{1}{N}\sum_{i=0}^{n-1}(i-\mu)^4}{\left[\frac{1}{N}\sum_{i=0}^{n-1}(x_i i)^2 - \mu^2\right]^{2}} - 3. \tag{11}$$

Classification of the holes is done by a simple neural network, trained by set of 300 exemplary vectors, each vector is composed of histogram parameters resulting from equations 7 - 11. Each exemplary vector is manually classified and this classification is verified by an experienced radiologist. Final segmentation of bones in complex structure of basis cranii can be seen in Fig. 4c.

5 EXTRACTION OF PARAMETERS FOR STATISTICAL ANALYSIS

By means of the segmentation algorithm proposed in section 4 two binary masks is obtained representing bones and soft tissue. In order to compare noise properties of images reconstructed by the iDose with respect to the conventional FBP technique, anatomical structures must be removed. Removing of anatomical structures is done by subtraction of image reconstructed by the FBP from images reconstructed by the iDose (i.e. images marked by ID30, ID50, ID70 and ID70MR) using binary masks for bones and soft tissue. Results of these subtractions are called residual noise images which can be seen in Fig. 6.

Standard deviation of residual noise

First group of parameters extracted from residual noise images are standard deviations computed from a whole image volume (results can be seen in Fig. 6). Meaning of this parameter is explained considering following thought. A region of interest (ROI) is selected from each reconstructed image (before subtraction) in order

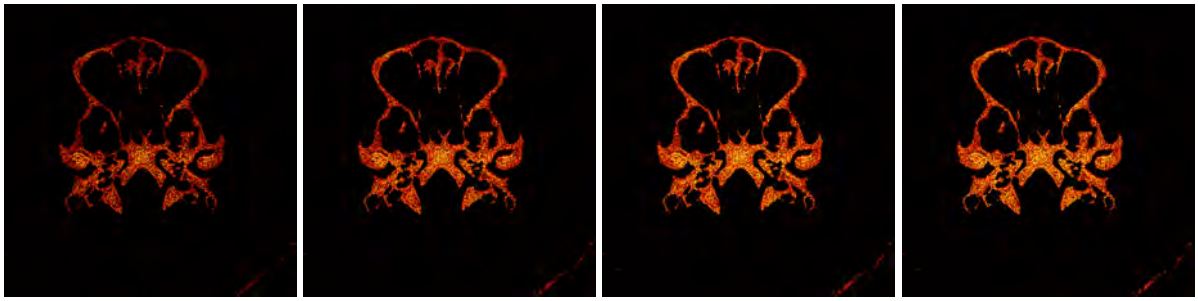


Figure 6: Residual noise images depicted separately for bones (upper row) and soft tissue (bottom row)

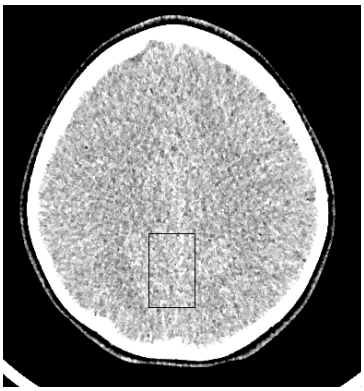


Figure 7: ROI selected from image reconstructed by ID70

to select an area of brain, which should be considered homogeneous. Therefore intensity changes in these ROIs are considered being only random noise patterns, see sample in Fig. 7.

Two parameters are computed from each ROI, a cross covariance with the ROI selected form the FBP reconstruction ($C_{X,FBP}$) and a standard deviation in each ROI (σ_X^2), see Tab. 1 and note that both parameters decreases with increasing iDose level. Subtracting of two random variables FBP and X (X is meant as a particular reconstruction), having variances σ_{FBP}^2 and σ_X^2 and cross-covariance $C_{X,FBP}$, results in new random variable with a standard deviation equal to equation (12).

$$\sigma_{(X-FBP)} = \sqrt{\sigma_{FBP}^2 + \sigma_X^2 - 2.C_{X,FBP}} \qquad (12)$$

Recon. (X)	$C_{X,FBP}$	σ_X^2	$\sigma_{(X-FBP)}$	$\sigma_{E(X-FBP)}$
FBP	276.64	16.63	0.41	0
ID30	234.09	14.12	2.78	2.77
ID50	200.31	12.19	4.95	4.95
ID70	158.81	9.97	7.64	7.64
ID70MR	155.11	9.46	7.47	7.48

Table 1: Noise parameters of selected ROI

A standard deviation of residual noise in investigated ROI $\sigma_{E(X-FBP)}$ is computed from real data and compared with value obtained from equation (12), see Tab. 1. Assuming that anatomical structures are identical in reconstructed images and completely suppressed by subtraction, the standard deviation of residual noise, therefore depends only on a standard deviation of noise in image X and a cross-covariance function between noises in images FBP and X. The standard deviation of residual noise increases with decreasing cross-covariance $C_{FBP,X}$ and increasing difference between σ_{FBP}^2 and σ_X^2 and thus can be considered as a valuable measure indicating improvement of noise properties in images reconstructed by the iDose according to images reconstructed by the filtered back projection. Advantage of this parameter lies in independence on an imaged object and therefore can be directly applicable to real patient data not only to phantoms. On the other hand it provides only relative improvement of noise properties according to the filtered back projection.

By means of the segmentation algorithm proposed in section 4 two binary masks is obtained representing bones and soft tissue. In order to compare noise properties of images reconstructed by the iDose with respect to the conventional FBP technique, anatomical structures must be removed. Removing of anatomical structures is done by subtraction of image reconstructed by the FBP from images reconstructed by the iDose (i.e. images marked by ID30, ID50, ID70 and ID70MR) using binary masks for bones and soft tissue. Results of these subtractions are called residual noise images which can be seen in Fig. 6

Power spectrum of residual noise

Standard deviation provides information only on noise magnitude, however no less important is knowledge about its frequency content. Such an information may be obtained by a noise power spectrum, routinely used as quality measure of MDCT imaging systems [YKH+08], [YKH+08] and [BMG07]. In this study residual noise images (Fig. 6), both for segmented bones and soft tissue, serves as input images for a noise power spectral analysis. Determination of a noise power spectra (NPS) is carried out by a direct digital technique as proposed in [SCJ02] and is computed according to equation (13).

$$S(f_x, f_y) = \frac{b_x b_y}{L_x L_y} \cdot \left\langle \left| DFT_{2D}\left\{ D(x,y) - D_{filt}(x,y) \right\} \right|^2 \right\rangle \tag{13}$$

Each slice is considered to be the one realization of a random noise and is denoted as $D(x,y)$. Individual realizations must be zero mean detrended before NPS calculation, therefore an image filtered by a lowpass Gaussian filter ($D_{filt}(x,y)$) is subtracted from each slice. Applying a two-dimensional Fourier transform and squaring an absolute value of a result ($|\circ|^2$), individual noise power spectrum is obtained. Individual noise power spectra suffer from a very large variance between realizations, power spectrum of a stochastic field (i.e. a process generating random noise) is therefore calculated as a mean value of individual power spectra (in equation (13) outlined by $\langle\circ\rangle$ operator). Fraction in this equation is a normalization term consisting of b_x b_y representing sampling periods and L_y L_x representing sizes in directions x and y, respectively.

Residual noise power spectra are determined in transverse $S(f_x, f_y)$, coronal $S(f_x, f_z)$ and sagittal plane $S(f_y, f_z)$ as can be seen in Fig. 8. A set of an annular sector shaped binary frequency filters, covering in piecewise sense the whole spectrum, is used to extract the final parameters from residual noise power spectra. Filters are used to select a segment of a NPS and the mean of this segment is the sought noise pattern, 36 parameters are extracted from each single residual noise power spectrum.

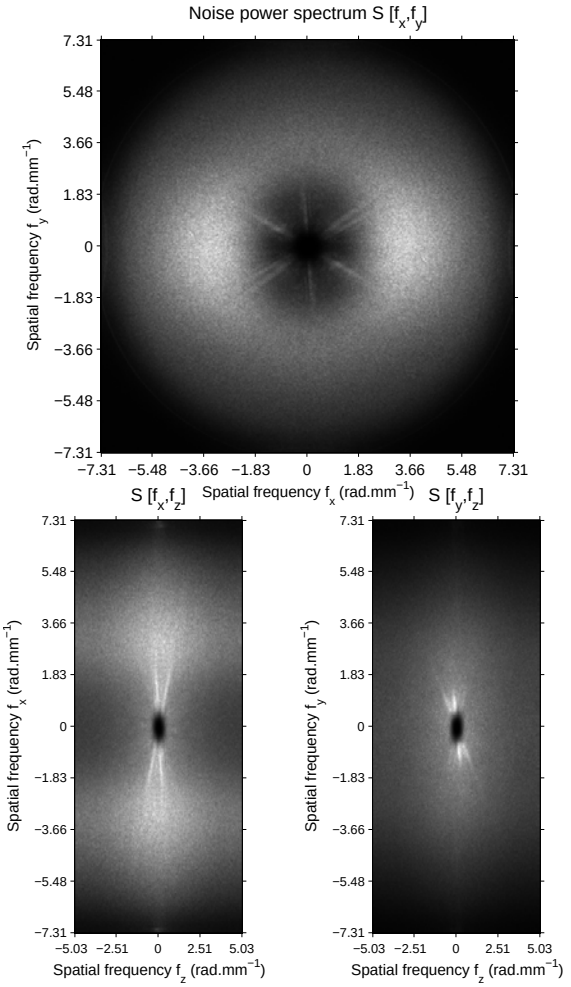


Figure 8: Example of power spectra of residual noise for transverse (upper), coronal (bottom left) and sagittal (bottom right) planes

6 CONCLUSIONS AND FUTURE WORK

Preprocessing of reconstructed image data and extraction of parameters for further statistical analysis of noise in MDCT images reconstructed by the iDose iterative algorithm is proposed in this paper. The preprocessing includes a correction of the stair-step artifact, which may harm further noise feature extraction and segmentation of bones and soft tissue. The proposed algorithm for segmenting bones in head images is fast and reliable even in complex structure of basis cranii, however there are certain drawbacks of this method. Segmentation of cortical bones, especially the ones with weak borders, may not result in areas of zeros fully surrounded by ones in locations of trabecular bones. Therefore boundary tracking algorithm can not label them as a "holes" and such a trabecular

bones remains unsegmented. Another difficulty is the lack of a trabecular structure in bones, especially in images of pediatrics patients. Considering that a trabecular structure in bones causes difference in the shapes of histograms of holes, the lack of this structure can negatively influence reliability of the resulting segmentation.

The parameters used for further statistical analysis are the standard deviation and noise power spectrum of the residual noise. The images formed by the residual noise are obtained by subtracting the images reconstructed by the filtered back projection from the images reconstructed by the iDose algorithm. When obtained by this subtraction, the noise properties can be evaluated in the whole volume of real patient data, on the other hand, the obtained parameters do not reflect the absolute level of the image noise but only the relative improvement with respect to image reconstructed by the FBP.

In order to assess different nature of noise and prove different behavior of the iterative reconstruction in soft tissue and bones the images of residual noise are multiplied with the binary masks obtained by segmentation. According to a convolution property of Fourier transform multiplying of signals results in convolution of their spectra therefore each of the residual noise power spectrum is affected by spectrum of the used binary mask which is moreover varying with respect to the slice position. Statistical inference of general results from such modified spectra may be incorrect and our future goal will be to analyze how strong is this influence and how to overcome this problem.

Considering the separate evaluation of the noise parameters from bones and soft tissue, taking into account the number of iDose reconstructions and the count of parameters extracted from three residual noise power spectra (for transverse, coronal and sagittal plane), we obtain 392 noise parameters per patient. The vectors of the parameters for forty patients can be arranged to matrix of size forty rows and 392 columns where each row can be considered as a single realization of a random process. Multidimensional statistical analysis such as principal component analysis or factor analysis can be used to reveal hidden relations in this matrix. Statistical analysis of the whole matrix can be rather complicated due to high number of the extracted parameters in comparison with quantity of scanned patients therefore selections of groups of parameters must be done (for example selection of low frequency noise). Results of future statistical analysis are expected to clarify relation between dose reduction, iDose level and quantity of image noise and differences between noise properties in soft tissue and bones. In future research proposed algorithms will be adapted to abdominal and thoracic images and typical noise properties of these body parts will be analyzed.

7 ACKNOWLEDGMENTS

Lending of the iDose reconstructor prototype programme package from Philips Healthcare is highly acknowledged as well as the long term data acquisition enabled by the Brno Faculty Hospital, Children's Hospital.

8 REFERENCES

[BK04] Julia F. Barrett and Nicholas Keat. Artifacts in CT: recognition and avoidance. *Radiographics : a review publication of the Radiological Society of North America, Inc*, 24(6):1679–91, 2004.

[BKK12] Marcel Beister, Daniel Kolditz, and Willi A. Kalender. Iterative reconstruction methods in X-ray CT. *Physica medica : PM : an international journal devoted to the applications of physics to medicine and biology : official journal of the Italian Association of Biomedical Physics (AIFB)*, 28(2):94–108, April 2012.

[BMG07] K. L. Boedeker and M. F. McNitt-Gray. Application of the noise power spectrum in modern diagnostic MDCT: part II. Noise power spectra and signal to noise. *Physics in medicine and biology*, 52(14):4047–61, 2007.

[FB11] Dominik Fleischmann and F. Edward Boas. Computed tomography–old ideas and new technology. *European radiology*, 21(3):510–7, March 2011.

[Goo12] Hyun Woo Goo. CT Radiation Dose Optimization and Estimation: an Update for Radiologists. *Korean journal of radiology : official journal of the Korean Radiological Society*, 13(1):1–11, January 2012.

[KH75] C. D. Kuglin and D. C. Hines. The phase correlation image alignment method. *IEEE Conference on Cybernetics and Society*, pages 163–165, 1975.

[MGB+12] Frédéric A. Miéville, François Gudinchet, Francis Brunelle, François O. Bochud, and Francis R. Verdun. Iterative reconstruction methods in two different MDCT scanners: Physical metrics and 4-alternative forced-choice detectability experiments - A phantom approach. *Physica Medica*, January 2012.

[MNS+10] Daniele Marin, Rendon C. Nelson, Sebastian T. Schindera, Samuel Richard, Richard S. Youngblood, Terry T. Yoshizumi, and Ehsan Samei. Low-tube-voltage, high-tube-current multidetector abdominal CT: improved image quality and decreased radiation dose with adaptive statistical iterative reconstruction algorithm–initial clinical experience. *Radiology*, 254(1):145–53, January 2010.

[MPB+09] Cynthia H. McCollough, Andrew N. Primak, Natalie Braun, James Kofler, Lifeng Yu, and Jodie Christner. Strategies for reducing radiation dose in CT. *Radiologic clinics of North America*, 47(1):27–40, January 2009.

[SCJ02] J. H. Siewerdsen, I. A. Cunningham, and D. A. Jaffray. A framework for noise-power spectrum analysis of multidimensional images. *Medical Physics*, 29(11):2655, 2002.

[YKH+08] Kai Yang, Alexander L. C. Kwan, Shih-Ying Huang, Nathan J. Packard, and John M. Boone. Noise power properties of a cone-beam CT system for breast cancer detection. *Medical Physics*, 35(12):5317, 2008.

Real-time Volume Rendering and Tractography Visualization on the Web

John Congote[1], Esther Novo, Luis Kabongo
Vicomtech Research Center
Donostia - San Sebastian, Spain
jcongote,enovo,lkabongo@vicomtech.org

Dan Ginsburg
Children's Hospital
Boston, United States
dginsburg@upsamplesoftware.com

Stephan Gerhard
Institute of Neuroinformatics
Uni/ETH Zurich, Switzerland
connectome@unidesign.ch

Rudolph Pienaar
Harvard Medical School
Boston, United States
Rudolph.Pienaar@childrens.harvard.edu

Oscar E. Ruiz
[1]Universidad EAFIT
Medellin, Antioquia
oruiz@eafit.edu.co

ABSTRACT

In the field of computer graphics, *Volume Rendering* techniques allow the visualization of 3D datasets, and specifically, Volume Ray-Casting renders images from volumetric datasets, typically used in some scientific areas, such as medical imaging. This article aims to describe the development of a combined visualization of *tractography* and *volume rendering* of brain T1 MRI images in an integrated way. An innovative web viewer for interactive visualization of neuro-imaging data has been developed based on *WebGL*. This recently developed standard enables the clients to use the web viewer on a wide range of devices, with the only requirement of a compliant web-browser. As the majority of the rendering tasks take place in the client machine, the effect of bottlenecks and server overloading are minimized. The web application presented is able to compete with desktop tools, even supporting high graphical demands and facing challenges regarding performance and scalability. The developed software modules are available as open source code and include MRI volume data and tractography generated by the Diffusion Toolkit, and connectivity data from the Connectome Mapping Toolkit. Our contribution for the Volume Web Viewer implements early ray termination step according to the tractography depthmap, combining volume images and estimated white matter fibers. Furthermore, the depthmap system extension can be used for visualization of other types of data, where geometric and volume elements are displayed simultaneously.

Keywords

WebGL, Volume Rendering, Ray Casting, DVR, dMRI

1 INTRODUCTION

Three-dimensional data can be found in several scientific fields, coming from simulation, sampling or modeling processes. Regarding the biomedical scope, several scanning techniques, such as magnetic resonance (MRI) or computerized tomography (CT), are used for storing body imaging samples as volumetric datasets formed by groups of parallel slices, where the term volumetric dataset refers to a scalar field. These datasets are usually visualized in three dimensions in order to facilitate specialists to interpret information.

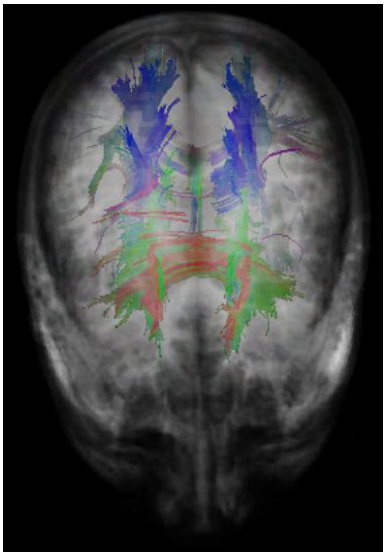


Figure 1: Combined visualization of volume rendering and tractography information on the web

Visualization of medical volumetric datasets can suitably be performed by the use of *Direct Volume Rendering* algorithms. These methods show important characteristics of datasets, even though rendering is not usually photo-realistic.The problem addressed in this paper is the visualization of tractography information obtained from dMRI (diffusion MRI) together with volume data corresponding to MRI or CT images.

In order to represent the volumetric datasets, volume rendering techniques allow the visualization of all inner characteristics of volumes at once, by projecting data into 2D images, according to the corresponding position of a virtual camera. The main idea of the ray-casting algorithm is to launch rays from the camera into the volume, calculating the volume rendering integral along the rays. Thus, in this method, the colour and opacity of each pixel in the final image is evaluated by launching a ray in the scene from the view position, sampling the volume at discrete points along the ray and accumulating the contribution of each sample.

Our contribution is an implementation of a web rendering system for medical images, which integrates volume rendering and geometric objects within a compliant WebGL browser, based on the volume ray casting algorithm and built on previous developments [CSK11]. Due to the technology limitations of Webgl, the improvements developed allow us to create a web application for combined visualization of volume rendering and tractography, as shown in Figure 1, being able to compete with desktop tools, supporting high graphical demands and facing challenges regarding performance and scalability.

The article is organized as follows. Section 2 presents the work related to this article, including a description of volume rendering techniques, visualization of medical images and geometry intersection. The methodology of the developed work is explained in Section 3. Then, the results accomplished are presented, and finally, Section 5 states the conclusions and future developments.

2 RELATED WORK

2.1 Volume Rendering

In computer graphics, Ray Casting is a well known direct volume rendering technique that was designed by Kajiya and Herzen [KVH84] as one of the initial developments in this area. Traditionally, three dimensional objects have been created by using surface representations, drawing geometric primitives that create polygonal meshes [Lev88], hence provoking the loss of information from one dimension.

Further developments [DCH88] accomplished the mathematical modeling of the ray casting process, based on the light's behaviour equations. Thus, the volume rendering integral was defined. A comparative between different direct volume rendering algorithms, such as Texture Mapping, Ray Casting, Splatting or Shear Warp, was presented [MHB00]. Ray casting is a flexible algorithm that allows the implementation of acceleration methods, such as Empty Space Skipping [KW03] or *Early Ray Termination*. Early ray termination is an optimization process that establishes certain limitations in the volume, so that the samples encountered after them do not contribute to the value of the pixel.

Ray casting suitably fits GPUs' operating mode [Sch05], because of the independence of each ray that is launched to the scene, making this algorithm highly parallelizable and allowing the exploitation of GPUs' parallel architecture. For GPU ray casting, the volume element is stored in the GPU memory as a 3D texture and a fragment shader program is used in order to implement the ray casting algorithm.

A quality evaluation model was developed for comparing the different Direct Volume Rendering techniques [BBD07]. These methods handle a higher amount of data than surface rendering techniques, therefore, the complexity of the algorithms is increased, as well as the necessary rendering time [Bru08]. Optimized volume rendering methods avoid empty spaces by introducing a volume proxy geometry [MRH08].

Web 3D Rendering

The use of the recently released WebGL standard [Mar11] leads to new methods for web 3D visualization, where most part of the computational processes are performed in vertex and fragment shaders that run on the GPU hardware. WebGL is a software library that enables HTML5-based browsers to identify clients' graphics hardware. HTML5, the latest Internet standard propose, provides native elements for audio and video. WebGL consists of a low-level imperative graphic programming API based on OpenGLES 2.0 for Javascript that enables flexibility and exploits the characteristics of advanced graphics cards. Due to the constant improvement of the performance of Javascript interpreters, the management of scene elements behaves similarly to the ones obtained by using natively compiled languages. Moreover, some WebGL extensions have been implemented in order to achieve a friendly interaction, such as SpiderGL [DBPGS10].

Several standards and proprietary solutions are currently being developed in order to fulfil the necessity of moving 3D visualization into the web [BA01], such as X3D, a standard derived from VRML that stores 3D information in a scenegraph format using XML (Extensible Markup Language). This model has been implemented in a declarative form, as an extension of HTML; X3DOM presents a framework for integrating

X3D nodes into HTML5 DOM content [BEJZ09] and other alternatives have also been developed, e.g. XML3D [SKR10]. Finally, there is a standardization for X3D in the MedX3D volume rendering model [JAC08, PWS11].

2.2 Visualization of Medical Images

Medical visualization is a challenging scientific field because interpretation of images may lead to clinical intervention. Therefore, quality and fast interactive response are important features in this domain. Remarkable advances have occurred in medical imaging technology and applications in the past few years, supporting the possibility of sharing imaging data online across clinical and research centres and among clinicians and patients. The development of these kind of applications is influenced by connectivity, security and resources' heterogeneity concerns.

On-server rendering can be considered a partial solution for Medical Imaging [BM07]. Moreover, several web implementations for volumetric visualization have already been presented [JAC08], although many of these solutions require third party systems to allow visualization or their scalability is limited by the rendering server.

As medical volumetric imaging requires high fidelity and high performance, several rendering algorithms have been analyzed, leading to thread- and data-parallel implementations of ray casting [SHC09]. Thus, architectural trends of three modern commodity parallel architectures are exploited: multi-core, GPU, and Intel Larrabee. Other approaches describe the development of web-based 3D reconstruction and visualization frameworks for medical data [SAO10]. Such applications based on X3D technology allow extending cross-platform, inter-application data transfer ability. Several applications have been implemented using web 3D rendering techniques, for example, evaluation systems at the educational level [Joh07] or medical training simulations [JROB08].

dMRI

Diffusion Magnetic Resonance Imaging (dMRI) relies on the visualization of water diffusion using data from MRI. Diverse methodologies have been presented over the last years and can be classified into two categories: Image based and Object based techniques. The first methodology divides the space in voxels and the assigned colour represents the principal diffusion direction [MAA03]. However, tracks can not be easily identified since no segmentation of the visualization is performed, and therefore direction information is difficult to observe since voxel colour mapping is not one-to-one, *i.e.*, different directions might be represented by the same colour. Otherwise, in object based methodologies, objects, such as ellipsoids and lines, are used together with colour mapping in order to enhance visualization and give a direction sense to the representation.

Visualization of brain white matter cortical tracks is one of the most important applications of dMRI, since it allows to non-invasively visualize white matter anatomy, and detecting of anomalies [NVLM07, GKN11]. Tractography, which refers specifically to the representation of the white matter tracks based on the water diffusion information, employs lines to represent the diffusion direction and to visualize the white matter paths. In general, lines are generated using randomly distributed seed points; together with the principal diffusion information and a prescribed interval of time, the different paths are generated. However, this representation becomes dependent on the amount and location of seed points to correctly visualize tracks [EKG06] because erroneous connections might be produced between tracks due to the existing error in data. Incorrect visualization of branching of tracks is another drawback, since only one path is generated per each seed point.

Probabilistic methodologies have been proposed [EKG06] to represent branching of white matter tracks, in which secondary seed points are included in regions in which branching is assumed. Therefore, a denser visualization is performed in those regions. An algorithm was proposed for path visualization [RSDH10], in which the different global paths are simplified by one simple curve, clustering the different paths and then using average curves to obtain one simple curve that summarizes each cluster.

2.3 Geometry Intersection

The application described in this article requires representing volume rendering and tractography together, *i.e.*, both volumetric and polygonal data have to be displayed in the same scene. There are several models for combining polygonal geometry and volume rendering. Some methods identify the intersection between rays launched in the volume rendering process and geometry [SMF00]. This technique can be optimized by creating octrees for dividing the geometric space and prove intersections correctly.

Other models try to achieve a correct visibility order for the intersections between volume and geometry [HLSR09]. Geometry has to be rendered in the first place to correctly look at the intersections of the geometry and the volume. Besides, parts that are occluded by the geometry should not contribute to the final image, not performing any ray casting at all. In order to achieve this feature, rays should terminate when they hit a polygonal object, accordingly modifying the ray length image if a polygonal object is closer to the view point than the initial ray length.

3 METHODOLOGY

In our project, the results of the Connectome Mapper are directly loaded in the browser using WebGL and JavaScript. The FreeSurfer cortical surface reconstruction binary files are loaded and processed in JavaScript and converted to WebGL vertex buffer objects for rendering. The surfaces are overlaid with per-vertex curvature values computed during the FreeSurfer processing stream. The tractography data is likewise parsed in the JavaScript code and rendered as line primitives coloured based on direction. Finally, the structural network itself is converted to JSON (JavaScript Object Notation) as an offline preprocess and loaded into the browser using JavaScript. The networks are visualized in 3D along with the fiber tracts and volumes enabling exploration of connectivity information in real-time.

The work described in this paper has been developed using volume ray casting, a widely used algorithm for generation of 2D representations from three dimensional volumetric datasets. The obtained images are 2-dimensional matrices $I : [1,h] \times [1,w] \rightarrow \mathbb{R}^4$ (w: width and h: height, both in pixels). Each pixel is represented by a colour expressed by a four-tuple of red, green, blue and alpha real-valued components, $(R,G,B,A \in [0,1])$.

An entire volume is represented by a 3-dimensional array of real values $V : [1,H] \times [1,W] \times [1,D] \rightarrow [0,1]$ (H: Height, W: Width, D: Depth of the represented volume, all of them in positive integer coordinates). Therefore, $V(x,y,z) \in [0,1]$. The projection model used in this work is called pin-hole camera [HZ03]. The pin-hole camera technique uses intrinsic $K \in M_{3\times4}$ and extrinsic $R \in M_{4\times4}$ real-valued matrices in order to project every 3D point $p \in \mathbb{P}^3$ onto a 2D point $p' \in \mathbb{P}^2$.

The volume ray casting algorithm defines the colour for each pixel (i,j) in the image, which is also known as projection screen, I, according to the values of a scalar field $V(x,y,z)$. This scalar field is associated to the points (x,y,z) reached by rays that are originated at a certain pixel or camera, represented as C in Figure 2. A cuboid geometry is generated with coordinates $(0,0,0)$ to $(1,1,1)$. This cube represents the boundary established for the volumetric dataset. Each ray intersects with the cuboid volume V at points $p_{(i,j)}(x,y,z)$ and $q_{(i,j)}(x,y,z)$, which represent the input and output coordinates of the ray into and out from the volume, respectively.

Then, each obtained ray pq is equi-parametrically sampled. For every sampled point $s(x,y,z)$ over the ray, an approximation of the scalar field $V(s)$ is calculated, commonly by using trilinear interpolation. The sampled points also influence the colour of the originating pixel, due to the use of a composition function (Equations 1-4), where the accumulated colour A_{rgb} is the colour of the point s in the volume V, and A_a is the transparency component of the pixel, which has a value of 1 at the end of the rendering process. Given a certain set of coordinates (x,y,z) in the volume and a ray step k, V_a is the scalar value of the volume V, V_{rgb} is the colour defined by the given transfer function V_a, S represents the sampled values over the ray and O_f, L_f are the general Opacity and Light factors.

$$S_a = V_a \times O_f \times \left(\frac{1}{s}\right) \tag{1}$$

$$S_{rgb} = V_{rgb} \times S_a \times L_f \tag{2}$$

$$A^k_{rgb} = A^{k-1}_{rgb} + \left(1 - A^{k-1}_a\right) \times S_{rgb} \tag{3}$$

$$A^k_a = A^{k-1}_a + S_a \tag{4}$$

In the ray casting process performed in this work, geometry G is formed by a set of segment lines L (although G could also be represented as a set of points P or triangles T). Each segment L is defined by two points in the space. Lines are projected through projection matrices onto a different image, where the values of colour (r,g,b,a) and depth $(depth)$ are defined for each pixel (x,y).

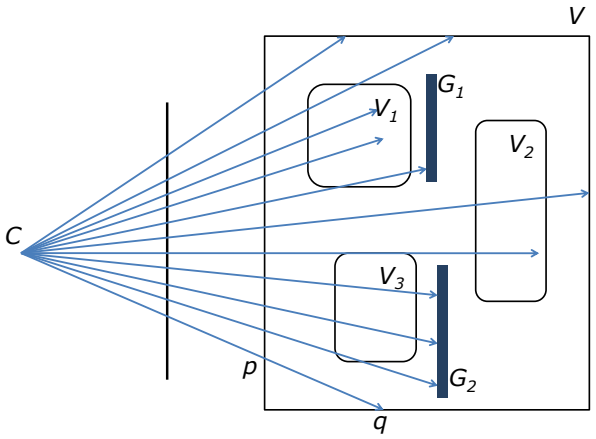


Figure 2: 2D representation of the ray casting algorithm performance (types of ray termination)

Each pq ray traverses the cuboid volume V, where both volume elements V_i and geometries G_i are rendered in the same process by modifying the early ray termination method, as depicted in Figure 2. This technique checks the alpha value for each sample of the transparency colour of the ray. If the value V_a is equal to 1, which means that the ray has reached the final colour, the remaining steps of the ray are not evaluated. Rays might terminate due to several reasons: when encountering a very dense volume (such as V_1 in fig. 2), when intersecting with a geometric element (*e.g.* with G_1) or when exiting the boundary cube, at point q.

The early ray termination model is also used to check the length of the ray and compare it to the depthmap of the figure. In conclusion, a projection of the geometry is

obtained, as well as the colour and depth for each pixel in the image. This information can be compared to the length of the ray, terminating the ray when the alpha value is 1 or when the depth is equal to the geometry depth.

4 RESULTS

This section describes the accomplished implementation of a real-time web viewer for both direct volume rendering and tractography visualization. This work is based on the WebGL standard and performs the ray casting algorithm with an early ray termination optimization.

4.1 Tractography

The Connectome Mapper [GDL11] is a publicly available software that provides a pipeline to automatically generate structural networks from raw dMRI data of the brain. Gray and white matter segmentations are obtained by processing T1 MPRAGE MRI using the Freesurfer set of tools. The Diffusion Toolkit is used later for reconstruction. A deterministic streamline algorithm is used to obtain tractography, by generating fiber tracts of the same subject. For cortical and subcortical regions of interest, a parcellation is performed. Finally, these datasets are coregistered and a network is generated by weighting the connectivity between regions based on the fiber tracts [GGCP11].

4.2 Data Processing and Volume Interpolation

For the developed work, all the slices that correspond to a certain volume are composed into a single image, as shown in Figure 3. This image is generated by placing slices in a matrix configuration as a preprocessing step of the rendering algorithm. The size of the image stored in GPU memory could range from 4096×4096 on a PC (which can contain up to 256^3 volume) to $1024x1024$ on other devices (which can contain up to $128 \times 128 \times 64$). The screen resolutions being reduced on mobile devices it seems reasonable to scale down or even crop the volumes original dimensions in order to match the maximum GPU available memory.

In medical imaging, the sample bit depth is usually higher than 8 bits per pixel. This is a drawback that has to be handled for the development of web applications, where commonly supported formats are limited to 8 bits per sample. In the described experiment, information from medical datasets was reduced to 8 bits per sample.

Identification of Ray Coordinates

According to the ray casting algorithm, the displayed colours of the boundary cuboid geometry represent

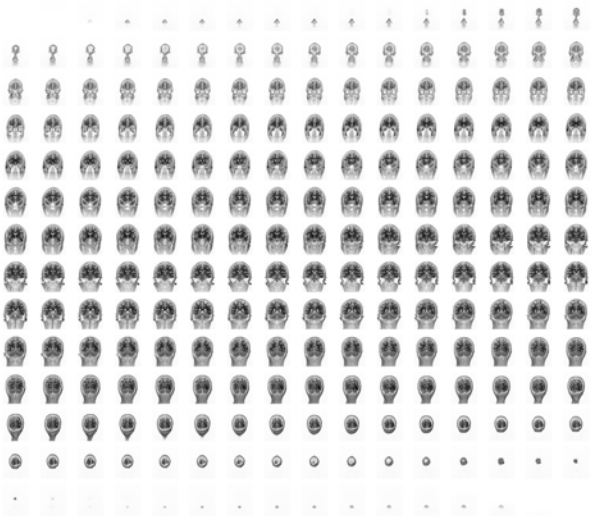


Figure 3: Brain dataset in mosaic form, read by the shader

the coordinates at each point (x,y,z). Coordinates are stored as r,g,b colour components for each pixel. Then, the cube can be rendered in the scene from the desired view point. In order to achieve volume visualization, several steps are followed in the rendering process. First of all, the rendering of the colour cube is performed according to the depth function change.

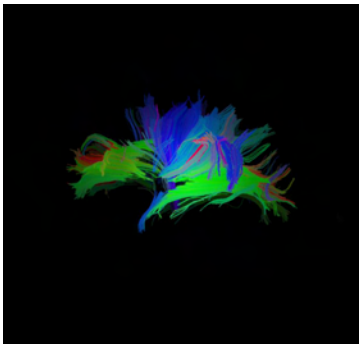
Taking this into account, rays are defined for each point of the cube, starting at the front faces, where the virtual camera is located, and ending at the back region. The colour of every point of the cube represents the exact coordinates of the ray for each pixel in the image. The colour information is stored as 24 bit RGB values. The range of values that can be represented may seem small or imprecise for large images, but colour interpolation provides precision enough for ray coordinates. The depth information is stored in different buffers in order to obtain the corresponding depth value for each ray. Finally, the geometry is rendered and the colour and depth buffers are stored to be processed in the volume shader.

4.3 Visualization

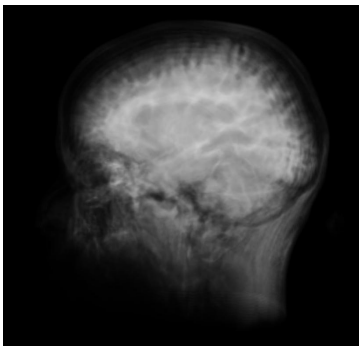
The previously presented GPU implementation of volume rendering based on WebGL was used to develop a real-time online tractography and volume rendering viewer, accordingly to Table 1, proving this standard to be a valid technology for real-time interactive applications on the web. The results shown in the table below were accomplished when interacting with the web viewer from several computers, using the same web browser (*Chrome*) and the same number of steps, 50. For every graphic card tested, the application can be completely considered to have a real-time behaviour.

Graphic card model	Frame rate
NVidia GeForce GTX480	60 fps
NVidia GeForce GTX285	60 fps
NVidia 9600GT	28 fps
NVidia Quadro FX 3800M	20 fps
NVidia Quadro FX 880M	15 fps

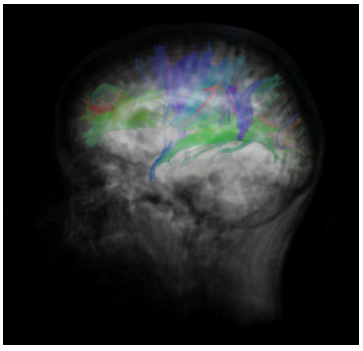
Table 1: Performance of the developed viewer for different graphic cards, using Chrome as web browser, the number of steps equal to 50



(a) Tractography



(b) Volume Rendering



(c) Combined visualization

Figure 4: Tractography, volume rendered image of brain T1 MPRAGE MRI and combined visualization on the web

In the developed work, the web viewer shows tractography information obtained from dMRI in the first place, represented in Figure 4(a). These organized fiber tracks in the white matter connect various cortical regions to each other. The tractography is represented using WebGL line primitives, where each fiber track is rendered by a set of points. The colour is assigned based on the absolute value of the unit vector pointing in the direction from the start point to the end point of the tract. The length value of each tract is stored in a per-vertex attribute together with the position and colour. The minimum tract length value is placed in a uniform variable in the vertex shader. The vertex shader determines whether the tract is longer than the minimum length to render. The entire tractrography set for the brain is efficiently rendered using a single draw call with one vertex buffer object. Thus, no dynamic geometry generation is performed in JavaScript.

Direct volume rendering of MRI data (Figures 4(b)) is developed simultaneously with the tractography. The volume renderer loads the MRI dataset from the server into a tiled 2D texture. Then, ray-tracing is performed in the shader in order to obtain the volume rendering. This implementation of a volume rendering system for the Web is based on the Volume Ray-Casting algorithm. Since the algorithm is implemented in WebGL, the reached visualization speed is similar to native applications, due to the use of the same accelerated graphic pipeline. The algorithm simulates 3D data by using a 2D tiling map of the slices from the volume maintaining trilinear interpolation and runs entirely in the client.

In the developed Web viewer, shown in Figure 5, the tractography and the volume rendering from brain MRI data can be represented separate or simultaneously, as depicted in Figures 4(c). Several features can be modified at runtime, by adjusting the provided sliders. Tractography's position can be changed according to the three main axes and fiber tracks can be seen more clearly by reducing the volume opacity. Finally, the minimum tract length can also be modified.

5 CONCLUSIONS AND FUTURE WORK

This paper describes the successful implementation of remote visualization of medical images based on WebGL[1]. Interaction with remote medical images was limited by many technical requirements, but the emergence of recent standards such as WebGL and HTML5 allow the development of applications that enable clients to access images without downloading them, maintaining

[1] http://www.volumerc.org/demos/brainviewer/webgl/brain_viewer/brain_viewer.html

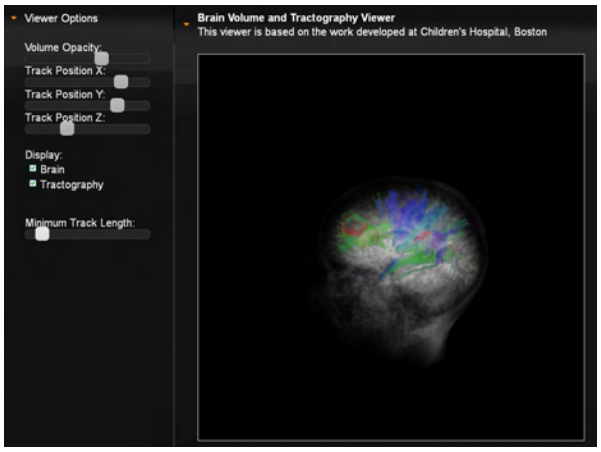


Figure 5: Volume rendering and tractography web viewer (sliders available for configuration)

data in a secure server and being able to perform functions, *e.g.* registration, segmentation, *etc.*, in a web context. These technologies empower web browsers to handle 3D graphics naturally. Thus, modern browsers support a wide range of applications, from simple rendering of two dimensional images to complex manipulation of 3D models.

The achieved visualization of volume rendering and tractography on the web, used for the implementation the presented viewer (shown in Figure 5), has demonstrated the capabilities of complex volume rendering visualization in web browsers, as well as the potential of WebGL for interactive visualization of neuroimaging data. Combined representation of volume rendering of brain T1 MRI images and tractography in real time has been accomplished. The main strength of the WebGL standard used here is the ability to provide efficient access to GPU rendering hardware with no special client-side software requirements, except for a compatible browser. Thus, this platform has great potential for imaging tools, particularly those providing web-based interfaces for automatic pipelining of image data processing.

In the work explained herein, the early ray termination algorithm was modified in order to combine volume and geometric elements in a seamless way. Thus, the developed software modules, which are available as open source code, successfully implement early ray termination step according to the tractography depthmap, performing a combination between volume images and estimated white matter fibers.

6 ACKNOWLEDGMENTS

This work was partially supported by CAD/CAM/CAE Laboratory at EAFIT University and the Colombian Council for Science and Technology -COLCIENCIAS-. Everyone who has contributed to this work is also gratefully acknowledged.

7 REFERENCES

[BA01] Johannes Behr and Marc Alexa. Volume visualization in vrml. In *Proceedings of the sixth international conference on 3D Web technology*, Web3D '01, pages 23–27, New York, NY, USA, 2001. ACM.

[BBD07] Christian Boucheny, Georges-Pierre Bonneau, Jacques Droulez, Guillaume Thibault, and Stéphane Ploix. A perceptive evaluation of volume rendering techniques. In *Proceedings of the 4th symposium on Applied perception in graphics and visualization*, APGV '07, pages 83–90, New York, NY, USA, 2007. ACM.

[BEJZ09] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[BM07] Bojan Blazona and Zeljka Mihajlovic. Visualization service based on web services. *29th International Conference on*, pages 673–678, 2007.

[Bru08] S. Bruckner. *Efficient Volume Visualization of Large Medical Datasets: Concepts and Algorithms*. VDM Verlag, 2008.

[CSK11] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar 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.

[DBPGS10] Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli, and Roberto Scopigno. Spidergl: a javascript 3d graphics library for next-generation www. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 165–174, New York, NY, USA, 2010. ACM.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, pages 65–74, New York, NY, USA, 1988. ACM.

[EKG06] H.H. Ehricke, U. Klose, and W. Grodd. Visualizing mr diffusion tensor fields by dynamic fiber tracking and uncertainty mapping. *Computers & Graphics*, 30(2):255–264, 2006.

[GDL11] S. Gerhard, A. Daducci, A. Lemkaddem, R. Meuli, J.P. Thiran, and P. Hagmann. The connectome viewer toolkit: an open source framework to manage, analyze, and visualize connectomes. *Frontiers in Neuroinformatics*, 5, 2011.

[GGCP11] Daniel Ginsburg, Stephan Gerhard, John Edgar Congote, and Rudolph Pienaar. Realtime visualization of the connectome in the browser using webgl. *Frontiers in Neuroinformatics*, October 2011.

[GKN11] A.J. Golby, G. Kindlmann, I. Norton, A. Yarmarkovich, S. Pieper, and R. Kikinis. Interactive diffusion tensor tractography visualization for neurosurgical planning. *Neurosurgery*, 68(2):496, 2011.

[HLSR09] Markus Hadwiger, Patric Ljung, Christof R. Salama, and Timo Ropinski. Advanced illumination techniques for gpu-based volume raycasting. In *ACM SIGGRAPH 2009 Courses*, pages 1–166. ACM, 2009.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second edition, 2003.

[JAC08] N W John, M Aratow, J Couch, D Evestedt, A D Hudson, N Polys, R F Puk, A Ray, K Victor, and Q Wang. Medx3d: Standards enabled desktop medical 3d. *Studies In Health Technology And Informatics*, 132:189–194, 2008.

[Joh07] Nigel W. John. The impact of web3d technologies on medical education and training. *Computers and Education*, 49(1):19 – 31, 2007. Web3D Technologies in Learning, Education and Training.

[JROB08] Yvonne Jung, Ruth Recker, Manuel Olbrich, and Ulrich Bockholt. Using x3d for medical training simulations. In *Web3D '08: Proceedings of the 13th international symposium on 3D web technology*, pages 43–51, New York, NY, USA, 2008. ACM.

[KVH84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18:165–174, January 1984.

[KW03] 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.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8:29–37, May 1988.

[MAA03] Y. Masutani, S. Aoki, O. Abe, N. Hayashi, and K. Otomo. Mr diffusion tensor imaging: recent advance and new techniques for diffusion tensor visualization. *European Journal of Radiology*, 46(1):53–66, 2003.

[Mar11] Chris Marrin. *WebGL Specification*. Khronos WebGL Working Group, 2011.

[MHB00] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90. Citeseer, 2000.

[MRH08] Jörg Mensmann, Timo Ropinski, and Klaus Hinrichs. ccelerating volume raycasting using occlusion frustums. In *IEEE/EG Volume and Point-Based Graphics*, pages 147–154, 2008.

[NVLM07] P.G.P. Nucifora, R. Verma, S.K. Lee, and E.R. Melhem. Diffusion-tensor mr imaging and tractography: Exploring brain microstructure and connectivity. *Radiology*, 245(2):367–384, 2007.

[PWS11] Nicholas Polys, Andrew Wood, and Patrick Shinpaugh. Cross-platform presentation of interactive volumetric imagery. Departmental Technical Report 1177, Virginia Tech, Advanced Research Computing, 2011.

[RSDH10] N. Ratnarajah, A. Simmons, O. Davydov, and A. Hojjat. A novel white matter fibre tracking algorithm using probabilistic tractography and average curves. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010*, pages 666–673, 2010.

[SAO10] S. Settapat, T. Achalakul, and M. Ohkura. Web-based 3d visualization and interaction of medical data using web3d. In *SICE Annual Conference 2010, Proceedings of*, pages 2986–2991. IEEE, 2010.

[Sch05] Henning Scharsach. Advanced gpu raycasting. *Proceedings of CESCG*, 5:67–76, 2005.

[SHC09] Mikhail Smelyanskiy, David Holmes, Jatin Chhugani, Alan Larson, Douglas M. Carmean, Dennis Hanson, Pradeep Dubey, Kurt Augustine, Daehyun Kim, Alan Kyker, Victor W. Lee, Anthony D. Nguyen, Larry Seiler, and Richard Robb. Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures. *IEEE Transactions on Visualization and Computer Graphics*, 15:1563–1570, November 2009.

[SKR10] Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. Xml3d: interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 175–184, New York, NY, USA, 2010. ACM.

[SMF00] Marcelo Rodrigo Maciel Silva, Isabel Harb Manssour, and Carla Maria Dal Sasso Freitas. Optimizing combined volume and surface data ray casting. In *WSCG*, 2000.

A Survey on Methods for Omnidirectional Shadow Rendering

Jan Navrátil
inavrati@fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

Jozef Kobrtek
xkobrt00@stud.fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

Pavel Zemčík
zemcik@fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

ABSTRACT

This paper focuses on methods of rendering shadows cast by point light sources. The goal is to summarize advantages and disadvantages of methods based on shadow mapping. We compare the traditional approach that exploits cube maps with the Dual–Paraboloid mapping. All of the methods are implemented on the latest hardware and they exploit capabilities of current GPUs. We also implemented optimization techniques which decrease the computational time. We examine the time the methods spent in particular rendering passes and we evaluate their overall performance. Finally, we conclude the comparison with some recommendations for typical applications in which the methods of interest can be exploited. We also suggest some direction of future investigation.

Keywords: shadow mapping, rendering, GPU, performance, cube maps, Dual–Paraboloid mapping

1 INTRODUCTION

Shadows play very important role in modern graphics applications as they increase overall visual cue from a rendered image. The shadow mapping algorithm [Wil78] and the technique based on shadow volumes [Cro77] are the most popular techniques for adding shadows to 3D scenes.

A well known disadvantage of the shadow mapping algorithm is the limited resolution of textures which store the depth information. Furthermore, it is also difficult to render shadows cast from point light sources. The basic shadow mapping algorithm cannot cover the whole environment with a single texture and thus additional computations are required. Such additional computations decrease the performance especially in scenes with a complex geometry.

The technique based on shadow volumes can easily render shadows from point light sources with per pixel accuracy. However, a high fill rate rapidly reduces the computational performance even for moderate sized scenes. Even though some optimization approaches exist [LWGM04], interactive applications mostly use the shadow mapping algorithm.

In this paper, we investigate several approaches for rendering shadows cast from point light sources based on the shadow mapping algorithm. Our contribution is the evaluation of the advantages and disadvantages of the approaches. We compare the existing methods especially with respect to their performance. We present some figures related to the time spent on generation of shadow maps on GPUs [MGR+05, Gru07] and also some frame times related to a camera view. We will also discuss the efficiency of all of the presented methods and potential implementation problems related to GPUs. Since the paper is restricted to the specific case of the shadow mapping algoritm we do not consider the shadow volumes approaches [LWGM04, VBGP09] as well as techniques that increase visual quality of shadows [WSP04]. Because they add some additional processing steps that might influence the results.

In Section 2, we refer to some techniques related to shadow rendering. We also mention some existing surveys. Section 3 introduces some issues that may arise when implementing the presented methods. We demonstrate all of the methods and their optimization in Section 4 and in Section 5, we present our experiments and discuss their results. We conclude our work in Section 6 where we also suggest some areas of future investigation.

2 RELATED WORK

For high quality shadow rendering, techniques such as ray tracing can be used. However, the shadow volumes algorithm or the shadow mapping approach are the most frequently used in interactive applications. The shadow volume technique [Cro77] provides per-pixel accuracy, its main disadvantage is a huge required fill rate. This fact does not allow for its common use in interactive applications. We can, however, find some methods that reduce the fill rate [LWGM04]. Nevertheless, the shadow mapping is the most popular algorithm for shadow ren-

dering. Basically, two approaches exist to render shadows cast from omnidirectional light sources using the shadow mapping algorithm. Firstly, shadow maps can be represented by faces of a cube map [Ger04]. In this case, six render passes are needed to fill the data into the cube map faces. Secondly, the Dual–Paraboloid mapping technique [BAS02, OBM06] can be used. It is capable of capturing the whole environment in two render passes. However, the mapping is not linear and thus not fully supported by contemporary graphics hardware. Recently, different techniques have been introduced [CVM11, HWL+11] that discuss other types of parametrizatons.

All of the above mentioned methods are capable of rendering shadows cast from omnidirectional (point) light sources. However, they all have some limitations and their usage may depend on the application and on scene complexity. Some surveys of shadow rendering have already been published, but they generally compare visual quality of the shadows with respect to the aliasing error [SWP10] or they address problem of soft shadow rendering [HLHS03]. In these cases, mostly directional light sources have been taken into account. The omnidirectional light sources need an extra treatment for creating shadow maps but also for reducing the aliasing error. Vanek et al. [VNHZ11] did some experiments with Dual–Paraboloid mapping technique but they did not work with the cube maps approach at all. They considered the cube map approach ineffective for omnidirectional light sources.

3 ISSUES OF THE SHADOW MAPPING ALGORITHM

3.1 Overview of the Algorithm

The first step of the shadow mapping algorithm is creation of the shadow map. A virtual camera is placed in the position of a light source. Then the scene is rendered as viewed from the virtual camera and the depth information is captured in the shadow map. In the second step, the scene is rendered from a camera point of view and the rendered pixels are compared with values stored in the shadow map.

During the process of the creation of the shadow map, the geometry has to be transformed to the light space coordinate system. For this purpose, the transformation matrix has to provide an appropriate transformation based on the type of the light source.

3.2 Linear Transformation and Interpolation

In case of directional light sources, orthogonal projection is used since all of the light rays have the same direction. For spotlights, perspective projection is used since the spotlights cover only certain part of the scene.

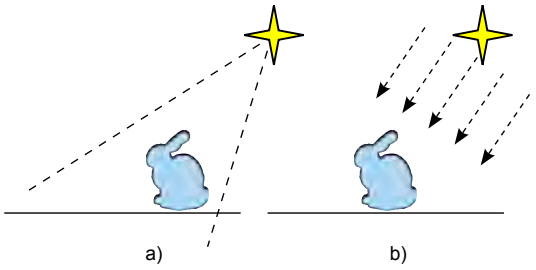


Figure 1: (left)A virtual camera for spotlights creates the view frustum. The frustum covers only a part of the scene based on a direction of the spotlight. (right) Directional lights use orthographic projection, because direction the light rays are parallel.

Then, the field-of-view in perspective projection is similar to the concept of falloff angle in spotlights. (see Figure 1). The perspective projection has a limited field-of-view range and thus it can not cover the whole environment. However, both projections are linear and thus they do not allow for covering the 180 degree field-of-view appropriately. To cover the whole environment, multiple linear projections are required. This means that if we want to use the basic shadow mapping algorithm for omnidirectional light sources, multiple render passes are necessary to create the shadow map (see Figure 2). Otherwise, a non-linear transformation has to be used.

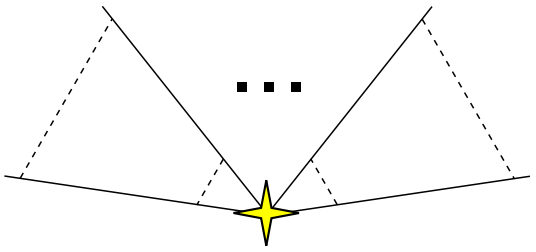


Figure 2: Multiple frusta have to be placed next to each other to cover the whole environment.

When we apply a projection, represented by a matrix, on vertices in the vertex shader, the fragments in the fragment shader are linearly interpolated. Instead of multiple linear projections, we can apply a non-linear transformation. The non-linear transformation, however, does not work well with the interpolation scheme used in graphics hardware. Straight lines are curved after the transformation (see Figure 3). It causes unwanted artifact for large polygons. The solution for these artifacts is to refine tessellation of the scene. For small polygons, the artifacts are not noticeable.

3.3 Limited Resolution

Stamminger et al. [SD02] described two types of aliasing: *perspective* and *projection*. Perspective aliasing is caused by limited resolution of shadow texture when

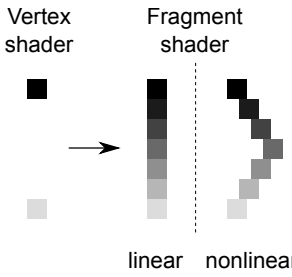


Figure 3: Fragments, that have to be rasterized between two vertices, are linearly interpolated in fragment shaders. Nonlinear parameterization can cause that fragments do not lie on a line.

the shadow map is undersampled while projection aliasing appears when the direction of light rays is parallel to the surface. Some methods exist that try to reduce the perspective aliasing artifacts on shadow boundaries. The shadow map can be filtered to make the shadow smooth [Fer05].

4 OVERVIEW OF METHODS

In this section, we present an overview of various methods for rendering shadows cast from omnidirectional light sources. We describe principles of each of the methods and we discuss their advantages and disadvantages. Furthermore, we present some optimization techniques that eliminate some of the disadvantages in order to achieve the best results for each of the methods. For our purpose, we only deal with omnidirectional light sources. It means that the light is emitted from a single point in space: therefore, we neglect an extent of the light source.

4.1 Cube Shadow Maps Technique

In Section 3, we mentioned how problematic it is to cover the whole environment with traditional projection transformations. In order to create shadow maps for an omnidirectional light source, it is necessary to point the virtual camera into six directions. The view direction of the virtual camera should point toward directions defined by the axes of the local coordinate system of the cube: positive X, negative X, positive Y, negative Y, positive Z and negative Z. This is almost identical to the way how a cube map for environment mapping is generated except that in this case depth values are stored instead of color.

Basics of the Cube Shadow Maps

The faces of the cube represent shadow maps and directions of the faces shows the particular direction for the virtual camera (see Figure 4). In order to cover the whole environment, the traditional shadow mapping algorithm exploits cube maps to visualize shadows cast from point lights. To fill the data in the cube shadow map, six render passes have to be performed. The GPUs generally support the cube shadow maps which are thus easy to implement.

The biggest disadvantage of the cube shadow maps is that six render passes are often too expensive. This fact can cause rapid decrease of performance for complex scenes with high number of polygons. Even if per-object frustum culling is applied, rendering of shadows is still very expensive compared to rendering of the rest of the scene.

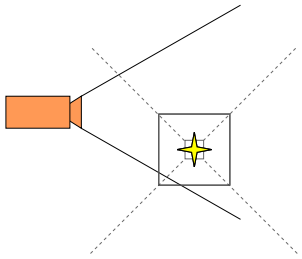


Figure 4: Illustration of the cube shadow maps technique. Each face of the cube stores depth values for a certain part of the scene.

Efficient Cube Face Frustum Culling

The methods of reduction of the number of passes have been investigated [KN05]. If the light source is outside the view frustum, then we can skip rendering of at least one face of the shadow map. This leads to the most noticeable effect on the performance

For our experiments, we use the following technique for efficient culling of cube map faces. A camera view frustum and each cube map frustum are tested for their mutual intersection. Those frusta that do not intersect can be discarded for further rendering because they do not affect the final image. The efficient culling of arbitrary frustum F against the camera view frustum V works as follows. The frusta are defined by 8 boundary points and 12 boundary edges. To determine whether the two frusta intersect, two symmetric tests have to be performed. Firstly, it should be tested whether a boundary point of one frustum lies inside other frustum (see Figure 5a). Secondly, it should be tested whether a boundary edge of one frustum intersects one or more clip planes of other frustum (see Figure 5b) [KN05].

For each face of the cube shadow map, we investigate whether the camera view frustum intersects the shadow face frustum and vice versa. If it is not the case, the shadow face frustum does not affect the scene and we can skip the additional processing (see Figure 6).

It is also necessary to take into account shadow casters outside the view frustum. If we cull the shadow caster against the view frustum, the projected shadow may still be visible in the view frustum. On the other

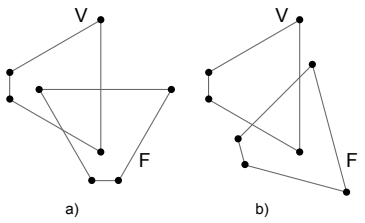


Figure 5: A frustum consists of boundary points and boundary edges. Two frusta intersect when (a) at least one boundary point of the frustum F lies inside other the frustum V or (b) at least one boundary edge of the frustum F intersects a face of the frustum V.

hand, culling the shadow caster against the cube map frustum draws invisible shadows as well. King et al. [KN05] suggest to use frustum-frustum intersection test described above for the shadow casters as well. Since we use point light sources, rays are emitted from a single point towards all shadow casters. This is analogous to the perspective projections. If the shadow casters are enclosed by bounding objects, frusta representing the projected shadows can be created [KN05] and then the frustum-frustum test can be applied in this case as well. These tests are performed once per frame.

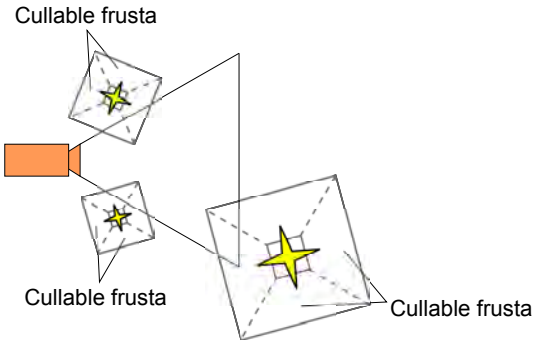


Figure 6: If the light source lies outside the camera view frustum, at least one face is cullable.

4.2 Dual–Paraboloid Mapping Algorithm

In the following text, we will discuss the Dual–Paraboloid Mapping algorithm (DPSM) [BAS02]. The mapping is based on two paraboloids attached back-to-back, each capturing one hemisphere:

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2+y^2), \quad x^2+y^2 \leq 1 \qquad (1)$$

In principle, a single hemisphere mapping can be imagined as an exploitation of a totally reflective mirror which reflects incident rays from the hemisphere into the direction of the paraboloid (see Figure 7). The rays may carry some information about the environment (mostly distance to the light) and the information can be stored into a texture. The texture coordinates are computed according to coordinates of the point where the ray is reflected. The Dual–Paraboloid mapping basically maps 3D space to 2D which is represented by the shadow map.

The algorithm needs only two render passes to capture the whole environment. Thus, it is more efficient than the cube shadow maps technique. Other parameterization can be certainly found (spherical, cube mapping etc.) but the proposed parabolic parameterization maintains its simplicity and performance, e.g. in GPU implementation [OBM06]. It minimizes the amount of used memory and the number of render passes that are necessary to cover the whole environment.

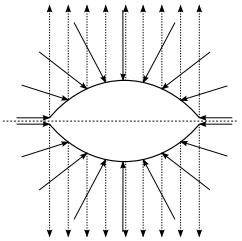


Figure 7: Two paraboloids attached back-to-back can capture the environment from all directions.

Nevertheless, the DPSM algorithm has also some disadvantages. While in the cube shadow map approach, all the transformations needed to create the shadow map are linear, they do not need any extra treatment on GPUs. This mainly concerns interpolation process between vertex and fragment shader (see Sec 3). When using the DPSM algorithm, the rendered scene needs to be finely tessellated because the mapping is not linear and thus it does not work well for large polygons. It may, however, introduce new bottlenecks.

4.3 Limitations of Geometry Shader

It is also possible to exploit both shadow mapping methods utilizing a geometry shader in order to reduce the number of render passes from six (two in Dual–Paraboloid mapping algorithm) to a single one [Eng08]. In this case, we exploited capablities of the frequently used graphics card, i.e., NVIDIA GeForce GTX560Ti, which supports geometry shaders.

The core of this method is usage of multiple render targets for and rendering all of the six cube map faces at once. The geometry shader transforms each of the incoming triangles with view-projection matrix of the corresponding cube. A naive approach sends all the incoming geometry data to all render targets, producing three to five times more geometry data than necessary. Such data is, however, anyhow discarded in the following rendering phases by the rasterizer. This leads to a massive performance penalty, as seen in Table 1. The results were measured on the same scene with the shadow map resolution set to 1024^2.

	avg. FPS
Cube6	6.19
Cube6Optim	20.3
DP	18.81
DPOptim	30.90

Table 1: All the methods exploit geometry shader and render the shadow maps in one pass.

This method was further optimized by testing each object bounding sphere against view frusta of the cube map faces, or, in case of Dual–Paraboloid mapping algorithm, against plane dividing scene in place of both paraboloids. Cube shadow mapping method was sped up by 227%, but still resulting in a very poor performance. Dual–Paraboloid mapping approach did not benefit that much from optimization, resulting in only 64% increase of performance, but also scoring far less than multi-pass methods.

Despite the optimizations, these methods did not overcome above mentioned optimized 6-pass techniques (described in Section 4.1). The core problem of the geometry shader is its execution model. It outputs data in a serial fashion with no parallelism used. Utilizing vertex shader and multiple passes overcomes the above mentioned geometry shader solutions despite switching of the render targets and updating resources between render calls.

5 EXPERIMENTAL RESULTS

We implemented the experimental framework in DirectX11 on an Intel Core i5 CPU 661 running at 3.33GHz using NVIDIA GeForce GTX560Ti GPU. The rendered images have resolution of 1024×768. We used the 32bit render target for the shadow maps. The resulting graphs were generated from an experimental walkthrough of a demo scene. The benchmarking scene had approximately 3 millions of vertices.

Our implementation does not introduce any hardware specific features. We can assume that the difference between the approaches would not be principally different.

5.1 Frame Time in Walkthrough

The first measurement shows dependence of the frame time for the walkthrough of the scene for all of the implemented methods. The unoptimized variants of the cube shadow maps and the Dual–Paraboloid shadow mapping (DPSM) show the worst results. In this approach, for every pass, all the geometry is rendered. Naturally, six render passes of the cube shadow maps lead into the highest frame time.

The basic optimization technique provided the bounding object frustum culling against the view frustum, the cube shadow maps frustum and the clipping plane for paraboloids. In this case, the same amount of geometry is rendered in both approaches. The overhead for increased number of the render passes for the cube shadow maps had no effect on an overall time for a single frame and thus the resulting frame times are similar.

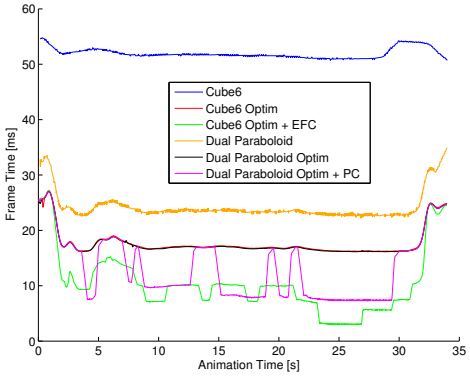


Figure 8: Frame times for the walkthrough of the scene for all implemented methods.

The cube shadow maps approach exhibits the best result with the effective cube face frustum culling - *EFC* (see Section 4.1). The plot shown in Figure 8 shows that the DPSM increased the performance only by skipping one paraboloid wherever appropriate (using plane clipping - *PC*). Otherwise, all of the geometry had to be rendered in two passes. The cube shadow maps approach can skip up to five render passes and thus it achieved the best results (e.g. in 25th second of the walkthrough). The frame time in the DPSM depends mainly on the amount of rendered geometry and also the amount of geometry in the given hemisphere. As can be seen in the plot, the DPSM saved only 50% of the computation time when rendering the scene only for one side. However, the cube shadow maps saved up to 83% of the performance. Furthermore, Figure 9 shows that the DPSM uses only one paraboloid most of the time and also that the cube shadow map rarely performed all six passes. This is mainly because the light source lied outside the camera view frustum.

5.2 Timings of Render Passes

Since the shadow mapping algorithm renders shadows in two passes, we investigated frame times for the passes for all implemented methods. The time for final shadow rendering showed to be equivalent for all methods, because it mainly depends on number of rendered geometry. Here, the view frustum culling was employed. The most noticeable differences were in times for generation of the shadow map.

As shown in Figure 10, the methods without any optimization had to render all the geometry six times in case of the cube shadow maps (blue) or two times in case of

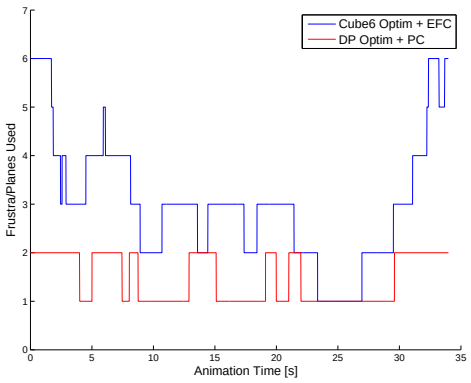


Figure 9: The plot shows the number of processed cube faces (blue) and the number of rendered paraboloid sides (red).

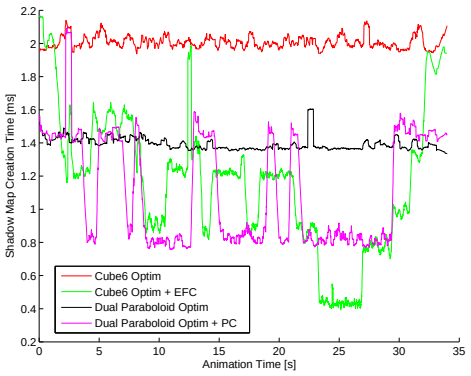


Figure 10: Evaluation of the times which all methods spent on the shadow map generation. For better illustration, unoptimized methods are not visible, because they had very poor results as compared to optimized techniques.

the DPSM algorithm (yellow). There are also some differences between methods where a frustum and plane culling is applied. The DPSM algorithm was faster compared to the cube shadow maps. An overall amount of rendered geometry was equivalent in both cases so there seems to be some additional overhead in the cube shadow maps technique.

Generally, the DPSM algorithm was faster when only one paraboloid was processed. The cube shadow map technique reached the similar times when only 2 faces were processed. The plot in Figure 10 also shows that in time 25 s, the cube shadow maps technique achieved the best results. In this case, only one face was processed which is mainly based on the position of a light sources relative to a camera (see Figure 11).

5.3 Effect of Shadow Map Resolution

We also investigated how the shadow map resolution affects the frame rate. In Table 2 and Table 3 you can see the results for various shadow map sizes. As you can see, the optimization techniques brought an increase in frame rate.

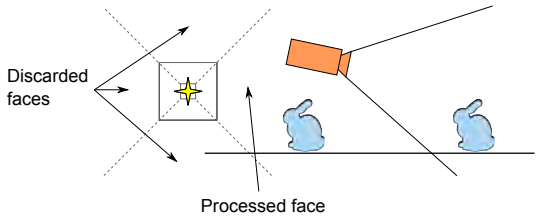


Figure 11: An illustration of the situation when only one face is processed during shadow map generation pass. Figure shows that only one cube face frustum intersects with the camera view frustum.

Considering shadow map as a texture storing single 32-bit value per texel, memory consumption of the cube shadow maps was from 24MB (1024×1024) to 384MB (4096×4096). Whereas the Dual–Paraboloid mapping approach uses one third of memory compared to the cube shadow maps (8MB to 128MB), it is more computationally intensive. Utilizing efficient frustum culling methods, we can save computation time by reducing number of the render passes and size of the geometry data, which also reduces memory utilization (less number of values stored due to frustum culling).

When taking 1024^2 resolution of shadow map as 100% performance for each method, switching to 2048^2 causes performance drop off only by 6.54% in average, but greatly increases shadow quality. Choosing 4096^2 resolution for shadow map takes 25.76% performance penalty in average.

Image quality of the result of Dual–Paraboloid mapping technique depends on the geometry of the occluding object. As described in [BAS02, OBM06], the Dual–Paraboloid mapping causes low-polygonal casters to produce incorrect shadows. Increasing shadow map resolution does improve shadow quality, but still can not match the quality of details achieved by the cube shadow maps approach (see Figure 12).

	1024^2	2048^2	4096^2
Cube6	75.71	70.04	47.9
Cube6Optim	150.43	116.76	64.04
Cube6Opt+EFC	188.71	151.67	89.68
DP	167.95	146.62	97.52
DPOptim	207.24	178.67	109.4
DPOptim+PC	208.15	180.24	110.95

Table 2: FPS of low-poly scene (600K vertices)

5.4 Position of a Light Source Relative to Geometry

We also performed an experiment where we focused on position of a light source relative to the geometry. This experiment was inspired by techniques for computation of interactive global illumination [RGK+08].

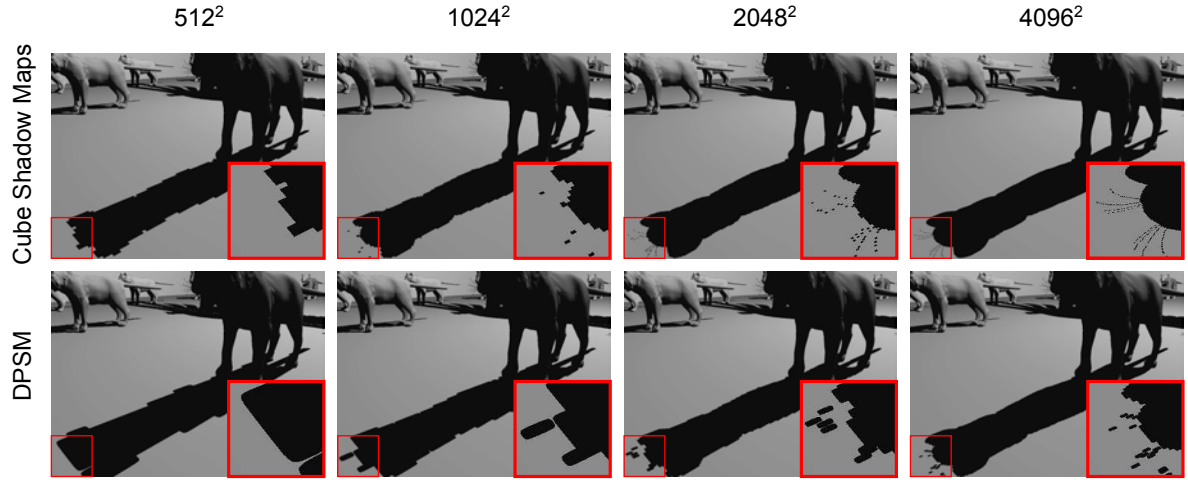


Figure 12: Figure shows how the shadow map resolution influences the shadow quality. Since a single paraboloid covers one hemisphere, one shadow map texel is projected on the large area in the scene (as compared to the cube shadow maps). This leads to worse quality of shadows.

	1024^2	2048^2	4096^2
Cube6	19.11	18.38	16.21
Cube6Optim	57.15	51.23	36.50
Cube6Opt+EFC	127.47	114.21	83.38
DP	41.50	39.74	33.17
DPOptim	57.47	54.32	42.85
DPOptim+PC	90.56	86.08	69.58

Table 3: FPS of high-poly scene (3M vertices)

In this case, *Virtual Point Lights* (VPLs) are generated on the surface to approximate indirect lighting. The reflected light is scattered into all directions. Therefore, some method is required to handle shadows from the reflected light. For this purpose, all the geometry data is positioned into one hemisphere relative to the light source. When the geometry is distributed around the light sources, it is useful to use the cube shadow maps technique, because it has better optimization strategy and it can easily manage the number of processed cube map faces. However, when we need to render only one hemisphere, the DPSM algorithm is more sufficient.

We measured times for generation of the shadow map in both of the presented techniques. Ritschel et al. [RGK+08] employed the Dual–Paraboloid mapping algorithm in their approach. They generated shadow maps for multiple VPLs (256 and more) from simplified geometry. We compared timings for the DPSM and the cube map technique.

In Figure 13, it can be seen that the DPSM algorithm is approximately two times faster than the cube shadow maps approach. The results are similar for various levels of the scene complexity. The Dual–Paraboloid mapping algorithm can be used despite its worse accuracy, because indirect lighting produces low-frequency shadows. In this case, the artifacts are blurred.

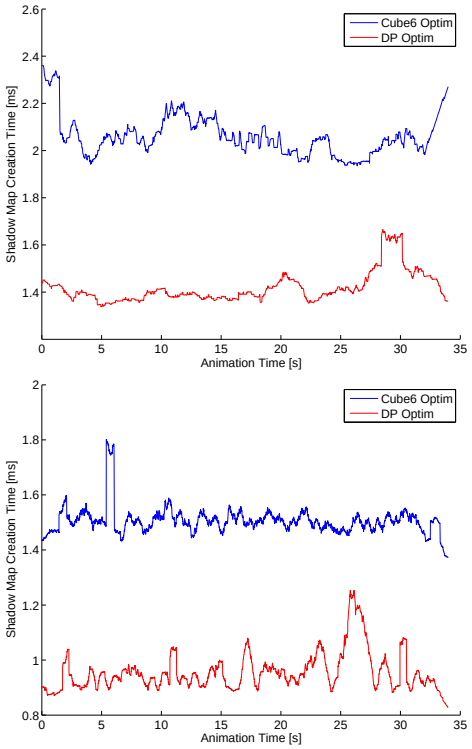


Figure 13: Figure illustrates times that the methods of interest spent on generation of the shadow map. In this case, the geometry is placed into one direction from the light source. The scene was represented by points only: 3 millions points (Top) and 100k points (Bottom).

6 CONCLUSION AND DISCUSSION

The goal of the work presented in this paper was to investigate the shadow mapping algorithm and techniques based on this algorithm as well as their capabilities to render shadows cast from point light sources. We ex-

amined two techniques that are based on the shadow mapping algorithm. The cube shadow maps approach exploits the traditional shadow mapping algorithm and renders the shadow map on cube faces. The Dual–Paraboloid shadow mapping uses nonlinear parameterization to render one hemisphere in one render pass.

The initial assumption was that multiple render passes performed by the cube shadow maps technique should be very time consuming process. The result of the measurement is that an unoptimized version of the cube shadow maps exhibits the worst performance of the examined algorithms. When a simple optimization technique was used significantly increased performance was reached, in fact, the best of the examined algorithms. The performance and the visual quality of the cube shadow maps is better compared to the Dual–Paraboloid algorithm. However, the Dual–Paraboloid algorithm produces better results if we consider the specific position of a light source related to a geometry, e.g., when computing illumination using VPLs.

Future work includes the complexity study that will improve the quality of measurements but since the timings depend mainly on the rendered geometry, however, as the complexity class is similar for all approaches, no significant differences are expected. It might be interesting to compare the implementation using the current hardware capabilities, e.g. CUDA. Evaluation of visual quality of the presented methods and their ability to deal with the aliasing problem in the shadow mapping algorithm is also subject of future work.

ACKNOWLEDGMENT

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070 and the Artemis JU project R3-COP, grant no. 100233.

REFERENCES

[AM00] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *J. Graph. Tools*, 5(1):9–22, January 2000.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Proceedings of Computer Graphics International*, pages 397–408, 2002.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.*, 11(2):242–248, 1977.

[CVM11] Marcel Stockli Contreras, Alberto José Ramírez Valadez, and Alejandro Jiménez Martínez. Dual sphere-unfolding method for single pass omni-directional shadow mapping. In *ACM SIGGRAPH 2011 Posters*, SIGGRAPH '11, pages 69:1–69:1, New York, NY, USA, 2011. ACM.

[Eng08] Wolfgang Engel, editor. *Programming Vertex, Geometry, and Pixel Shaders*. Charles River Media; 2 edition, 2008.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[Ger04] Philipp Gerasimov. Omnidirectional shadow mapping. In Randima Fernando, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, pages 193–203. Addison Wesley, 2004.

[Gru07] Holger Gruen. Performance profiling with amd gpu tools: A case study. AMD Sponsored Session, GDC, March 2007.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003.

[HWL+11] Tze-Yiu Ho, Liang Wan, Chi-Sing Leung, Ping-Man Lam, and Tien-Tsin Wong. Unicube for dynamic environment mapping. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):51–63, January 2011.

[KN05] Gary King and William Newhall. Efficient omnidirectional shadow maps. In Wolfgang Engle, editor, *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 435–448. Charles River Media, Hingham, MA, 2005.

[LWGM04] Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. Cc shadow volumes. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 146–, New York, NY, USA, 2004. ACM.

[MGR+05] Victor Moya, Carlos Gonzalez, Jordi Roca, Agustin Fernandez, and Roger Espasa. Shader performance analysis on a modern gpu architecture. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 355–364, Washington, DC, USA, 2005. IEEE Computer Society.

[OBM06] Brian Osman, Mike Bukowski, and Chris McEvoy. Practical implementation of dual paraboloid shadow maps. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 103–106. ACM, 2006.

[RGK+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8. ACM, 2008.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562. ACM, 2002.

[SWP10] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. In *EUROGRAPHICS 2010 State of the Art Reports*, 2010.

[VBGP09] Forest Vincent, Loïc Barthe, Gael Guennebaud, and Mathias Paulin. Soft Textured Shadow Volume. *Computer Graphics Forum*, 28(4):1111–1120, 2009.

[VNHZ11] Juraj Vanek, Jan Navrátil, Adam Herout, and Pavel Zemčík. High-quality shadows with improved paraboloid mapping. In *Proceedings of the 7th international conference on Advances in visual computing - Volume Part I*, ISVC'11, pages 421–430, Berlin, Heidelberg, 2011. Springer-Verlag.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978.

[WSP04] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps. *In the Eurographics Symposium on Rendering*, 2004.

TimeSeriesPaths: Projection-Based Explorative Analysis of Multivarate Time Series Data

Jürgen Bernard
Fraunhofer Institute for Computer Graphics Research, Darmstadt, Germany
juergen.bernard@igd.fraunhofer.de

Nils Wilhelm
Technische Universität Darmstadt, Germany
nwilhelm@rbg.informatik.tu-darmstadt.de

Maximilian Scherer
Interactive Graphics Systems Group, Technische Universität Darmstadt, Germany
maximilian.scherer@gris.tu-darmstadt.de

Thorsten May
Fraunhofer Institute for Computer Graphics Research, Darmstadt, Germany
thorsten.may@igd.fraunhofer.de

Tobias Schreck
Data Analysis and Visualization Group, Universität Konstanz, Germany
tobias.schreck@uni-konstanz.de

ABSTRACT

The analysis of time-dependent data is an important problem in many application domains, and interactive visualization of time-series data can help in understanding patterns in large time series data. Many effective approaches already exist for visual analysis of *univariate* time series supporting tasks such as assessment of data quality, detection of outliers, or identification of periodically or frequently occurring patterns. However, much fewer approaches exist which support *multivariate* time series. The existence of multiple values per time stamp makes the analysis task per se harder, and existing visualization techniques often do not scale well.

We introduce an approach for visual analysis of large multivariate time-dependent data, based on the idea of projecting multivariate measurements to a 2D display, visualizing the time dimension by trajectories. We use visual data aggregation metaphors based on grouping of similar data elements to scale with multivariate time series. Aggregation procedures can either be based on statistical properties of the data or on data clustering routines. Appropriately defined user controls allow to navigate and explore the data and interactively steer the parameters of the data aggregation to enhance data analysis. We present an implementation of our approach and apply it on a comprehensive data set from the field of earth observation, demonstrating the applicability and usefulness of our approach.

Keywords: Multivariate Time Series, Visual Cluster Analysis, Exploratory Data Analysis, Data Projection, Data Aggregation

1 INTRODUCTION

Multivariate time series data are gathered in many domains including economics, experimental physics, computer vision, robotics, and earth observation. E.g., in the financial domain, large amounts of stock prices are tracked over time; in earth observation, daily temperatures and many additional parameters are observed at specific locations over time; time-dependent measurements also arise in monitoring traffic parameters on a communication network. Analysis of time series data can take many forms, including assumption-free exploration; correlation of time series with each other; or evaluation of specific generative models. Much work has been done focused on analyzing one-dimensional time series, and respective solutions are often applied to multivariate data by analyzing each dependent variable versus an independent one. However, for multivariate data the widely used *IID* assumption (independent and identically distributed) usually does not hold. Therefore there is a need to analyze all dimensions of such data at once.

In the context of data mining and visual analytics, multivariate time series analysis is a difficult problem, with solutions typically relying, in some form or the other, on dimensionality reduction, feature selection, projection, and glyph-based visualization. The task at hand often includes finding periodic or frequent patterns in

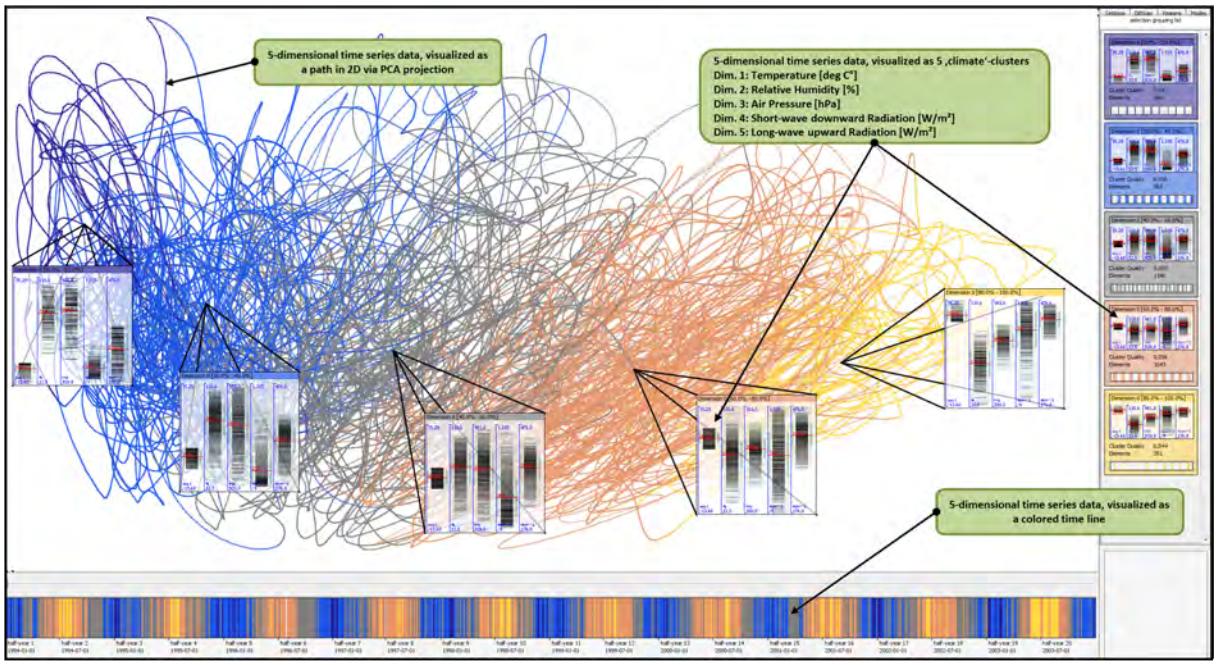


Figure 1: Main display: data analysis of a multivariate time series of 10 years length is always challenging due to overview problems and visual cluttering. This is the starting point of our data exploration. The Time Series Path System provides visual structures and interactive functionality to address the implied challenges. In this example, we aggregate a weather scenario by its temperature values and receive 5 well-distributed data clusters from cold (blue) on the left to warm (yellow) on the right. This is a qualified starting point for selection and filtering approaches to detect periodicity, dense data regions and outliers. Confer our case study in Section 5 for details about the 2D projection.

the data, relating multiple variables to each other, or detecting outliers or anomalies. Visual-interactive approaches can help to tackle these challenging tasks by closely involving the user in the exploration process, addressing the typically difficult parameter selection problem, which could be more complicated to solve relying on purely automatic methods.

Several works propose to visually analyze multivariate time-dependent data by dimensionality reduction [26, 12]. Multivariate data is visualized as two-dimensional time series paths obtained by dimensionality reduction (projection to 2D). While these works visually compare sections of labeled multivariate time-dependent data, they do not consider exploratory search in unknown data sets. Furthermore, these works do not focus on aggregation efforts to reduce over-plotting problems. To this end, we introduce interactively steerable data aggregation, supporting handling of multivariate time series data. In particular, the user is able to group data points according to data-specific characteristics like statistical calculations based on value and time, or clustering results.

Our approach supports an effective overview of frequent and infrequent *states* in multivariate time series data even in cases of very large data. Furthermore, users can interactively select meaningful path line subsets for detailed exploration and for visual clutter reduction purposes. Understanding of aggregated data groups is supported by showing a comprehensive cluster glyph metaphor, wherever data aggregation visualization is required within the exploration process. We directly involve the user in the exploration process, combining data exploration with interactive steering of the automatic analysis methods, such as searching for appropriate clustering parameters, in particular.

We demonstrate the usefulness of our approach by an application to earth observation data. There, long time series of many parameters arise, and users want to understand periodicities, trends, and anomalies. We show how our set of interactive views allows for interactively exploring weather patterns of different lengths and parameters. Due to our data aggregations, domain users can explore multivariate weather data in a single display, giving an overview of all data aspects at once.

The remainder of this paper is structured as follows. In Section 2 we discuss related work in several areas. In Section 3 and 4 we motivate our approach, explain our system design and describe user interaction techniques. In Section 5 we apply our implementation to a real-world data set, demonstrating the usefulness of the approach. Finally, we summarize this paper and discuss future extensions in Sections and 6 and 7.

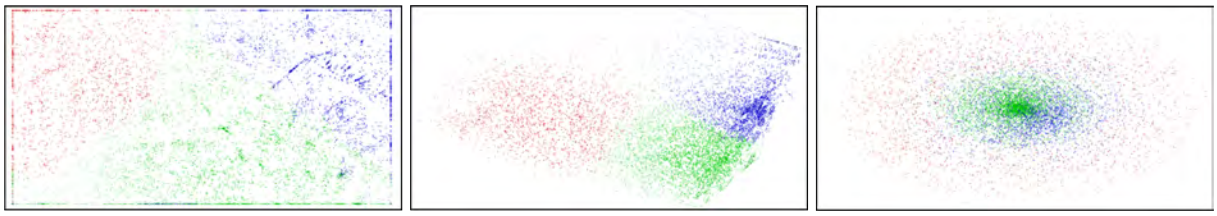


Figure 2: Visual comparison SOM, PCA and MDS projection technique. A k-means clustering result is shown.

2 RELATED WORK

Our work is related to analysis methods for time-dependent data and multivariate data. Time series analysis in general is conducted to understand the behavior of systems, to distinguish regular from extraordinary characteristics [14] and to predict future development [13].

Visualization of Time Series Data

The visualization of time series is often helpful for exploratory analysis. Traditionally, time series can be visualized by line charts [24]. However, using line charts is typically not effective for large time series data, as many and long time series lead to over-plotting if packed into a given display or would require excessive user navigation (cf. the problem specification in Figure 1). The *pixel paradigm* [2] for visualization of time series suggests to map the quantitative values of a time series to an appropriate color scale. Ultimately, each value can be represented by a single pixel. The Recursive Pattern technique [2] employs the pixel paradigm to arrange time series in a generic proximity-preserving way, allowing to arrange between row-by-row up to more complex patterns following space-filling curves. The comparison of many time series can be supported by rendering them next to each other in an appropriate display.

Besides side-by-side schemes, e.g., TreeMap-like layouts have been proposed [10]. An alternative to the pixel paradigm is to map the time axis to a spiral, effectively using more length, which is particularly useful for analysis of periodic data [27]. For domain-specific visualization tasks, e.g., atomistic simulation data, specialized techniques have been proposed [6]. An overview of time series visualization can be found in the textbook by Aigner et al. [1]

Automatic Support

Automatic analysis techniques are often used in time series visualization. E.g., the size of the data to be visualized may be reduced by aggregation [5] or dimensionality reduction [8].

In [25] prototypical time series patterns are found by cluster analysis, and linked to occurrence on the time scale by color-coding. In [17] a discretization approach is applied to support visual analysis of frequent subsequences in a node-link-diagram. Often, the similarity definition between time series or subsequences thereof is important to support exploratory search. In [28] so-called Perception Points of Interest are identified to sort a large number of time series for effective overviewing. Various other systems support the interactive retrieval of time series by defining appropriate similarity notions and query interfaces [9, 11, 3]. A visual-interactive approach to analyzing different families of functions is presented in [16]. Here, the authors allow the user to highlight data patterns of interest and provide linked views of the multidimensional data and the user-selected highlights.

Multivariate Time Series

The above methods mainly consider univariate time series. Yet, multivariate time series analysis is of importance in many domains. A number of approaches include small multiple displays for showing all variables over time next to each other. They may rely on line charts, pixel displays, or any other appropriate base technique. Also, automatic analysis methods for exploratory analysis in multivariate time series have been considered. E.g., in [19] a frequent-pattern-based approach is used to find interesting time series patterns along several levels of abstraction.

Recently, a number of authors have considered the visualization of multivariate time series data based on projection. The basic idea is to project discrete points in time to a 2D display, which in turn allows for analysis of the time series for regularities and irregularities [23]. In [22, 12] multivariate observation measures from motion tracking are projected using the Self-Organizing Map (SOM) method [15]. Individual observations are connected by lines, and glyphs illustrating the particular configurations of the motion are shown. In [18] multivariate time series are extracted from text, by computation of certain text features for discrete intervals along the sequence of the text. A PCA-based display was used to assess the development of the text content, by analysis of feature trajectories observed in the display. In [26] the authors use PCA-based projection to explore the sequence of small fixed-size intervals (so-called *n-grams*) of long univariate time series data. The approach was applied to stock market data and shown to provide an informative overview over long time series

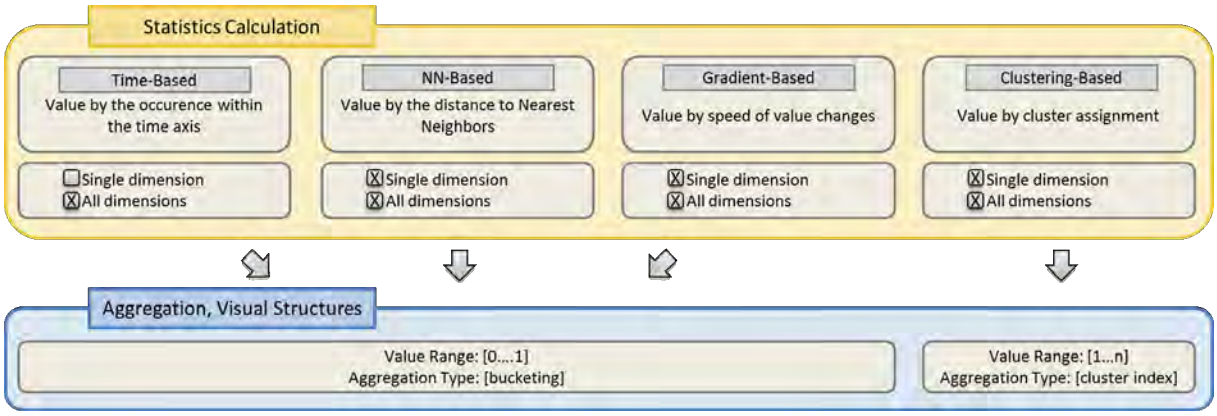


Figure 3: Aggregation of multivaritate time series data based on a variety of statistical data properties. Most of the functionality can either be performed on a single, or all dimensions.

data. In particular, the authors proposed their method to support the following analysis cases: Detection of cyclic behaviors; visual identification of base patterns and outliers; and analysis for trends.

3 SYSTEM DESIGN

In this work we present *TimeSeriesPaths*, a system for the analysis of multivariate time series data. The PCA projection algorithm is applied to arrange multivariate time-series on the (2D) display screen (the *Time Series Path Map*). We connect temporally adjacent data elements and receive a sequentially ordered set of points – a so called *time series path*. By default, such a visualization suffers from severe *over-plotting* and *overview problems*. In order to make such a visualization understandable for domain-experts and to counter the implied challenges, our approach comprises three contributions:

1. We apply semi-automatic data aggregation functionality, either derived from statistical data calculation, or from visual-interactive data clustering (cf. Subsection 3.2). This helps the user to get an *overview* to the dataset.

2. We present a cluster visualization technique that incorporates multiple information about the aggregated data (cf. Subsection 3.2). This supports *data interpretation and cluster comparison* approaches.

3. We propose a multi-view system with broad visual-interactive analysis functionality (cf. Subsection 4). Selection and highlighting modalities of data path subsets counter the challenge of *over-plotting* and allow for comprehensive *detail on demand* perspectives.

3.1 Visualizing Multivariate Time Series

Data Projection

We apply a projection technique to visualize multivariate time series data on 2D displays. An applicability consideration between visualizations based on projection and the multiple linechart technique is given in Section 6.

A general requirement concerning projection is the preservation of data topology, by means that similar data in the multivariate input space is also arranged close to each other in the display space. Due to their popularity and their diversity in arithmetical manner we chose PCA, SOM and Multidimensional Scaling MDS [7] as promising candidates. After an evaluation of our requirement criteria and a visual comparison in Figure 2, we choose the PCA algorithm as a default for prospective projection needs in the TimeSeriesPaths system. The non-linear MDS proves to be rather unsuitable for our approach, solely because it has troubles in separating k-means clusters. The SOM algorithm suffers in respect to the calculation speed and a major difficult (fully automatic) parametrization. Yet the key benefit of PCA derives from the ability to project data points in a linear manner, by means that the projection results do not lack on local distortions and thus allow for a more straight forwarded interpretation. Furthermore, the visual comparison of the three projection techniques shows a good cluster separation by PCA. We accept that PCA does not exploit the complete display space as well as the SOM projection. However later in this section, we will present our cluster glyph and show how our glyph visualization mitigates this problem.

Visualizing Time Series Paths

The visualization of time series paths is provided by the Time Series Path Map in the center of the display. Based on our data projection method, we connect individual data points by their chronological order to form paths. The projection arranges similar data points close to each other and reflects the data point distances of the multivariate data input space. Accordingly, if path sequences are similar to each other, their possibly close

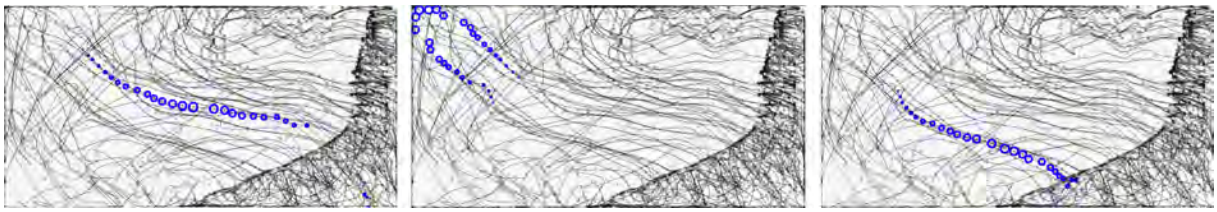


Figure 4: The "Rollercoaster Animation". By dragging the time slider, the user can explore the temporal development of the time series path. The cursor position and time series path neighbors are animated with circular shapes.

positions on the display space help the user with profound analysis approaches.

3.2 Multivariate Time Series Aggregation

Statistics Calculation and Aggregation

We integrate automatically generated statistical data information into the visualization to counter the overview problem and support the analysis process. So far, related approaches color-code data points for time-based and value-based changes or class labeling [12, 26]. Our approach generalizes this by a variety of statistical data measurements that provide additional important information, an overview is given in Figure 3. Altogether, our system provides four different properties of statistical data information for color coding:

- (a) occurrence within the time line (time-based)
- (b) nearest neighbor distance (NN-based)
- (c) speed of value change to adjacent time stamps (gradient-based)
- (d) cluster membership (clustering-based)

Except for (a), all statistical data information can either be calculated on a single dimension or on all dimensions of the data set. Thus, we are also able to perform domain-specific exploration tasks due to the level of detail in the aggregation setup. The number of data groups k can be specified by the user for all properties, (a)-(d).

Data color codings according to group affiliations are displayed on the Time Series Path Map, our time axis display at the bottom (called *Color Time Bar*), and the *Data Aggregation List* on the right, respectively. Showing multiple aspects of the data enables to find an appropriate aggregation level, to interpret groups of data and derive mutual characteristics, to detect outliers and to explore periodic behavior in the data.

In our case study (cf. Section 5), we will show that distributions of aggregated statistical data information on the Time Series Path Map and the Color Time Bar give valuable information about dense data regions, data anomalies and the periodicity of time series paths.

Generic Cluster Glyph

The aggregation of data into groups requires a meaningful cluster visualization method (cf. Figure 5). The main requirement is genericity in order to suit to a great variety of multivariate time series data. Additionally, averages, minima and maxima, variances, number of elements and cluster quality indices are needed. Each data dimension is displayed with an error bar chart glyph metaphor and labeled with the corresponding physical unit. Additionally, we include the distribution of time stamps on a time axis to monitor chronological data characteristics to detect periodic behavior or anomalies. Finally we demand the cluster glyph to show the cluster color coding for linking, and a headline for user-centered data labeling purposes.

Earlier we argued that PCA does not capitalize the entire border areas of the display space. We benefit from this instance due to the fact that we have free space remaining to position cluster glyphs for data aggregation operations. Four black concentrical lines connect the cluster glyph with the appropriate display coordinate without producing too much occlusion (see Figures 1, 6, 7 and 8).

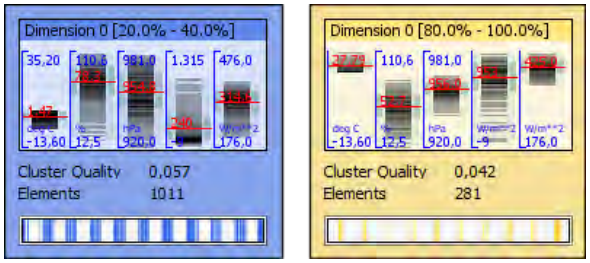


Figure 5: Generic Cluster Glyph. A boxplot-like visualization shows the distribution of data elements in each dimension of the dataset, transparency is used to show data frequency. Cluster centroid values are displayed as red bars, just like gray variance bands mapped laterally for each dimension. Statistical information about the data cluster is shown at the center, the data distribution on the global time axis is shown at the bottom.

4 INTERACTION TECHNIQUES

TimeSeriesPaths includes a set of linked user interaction modalities which work across the three different

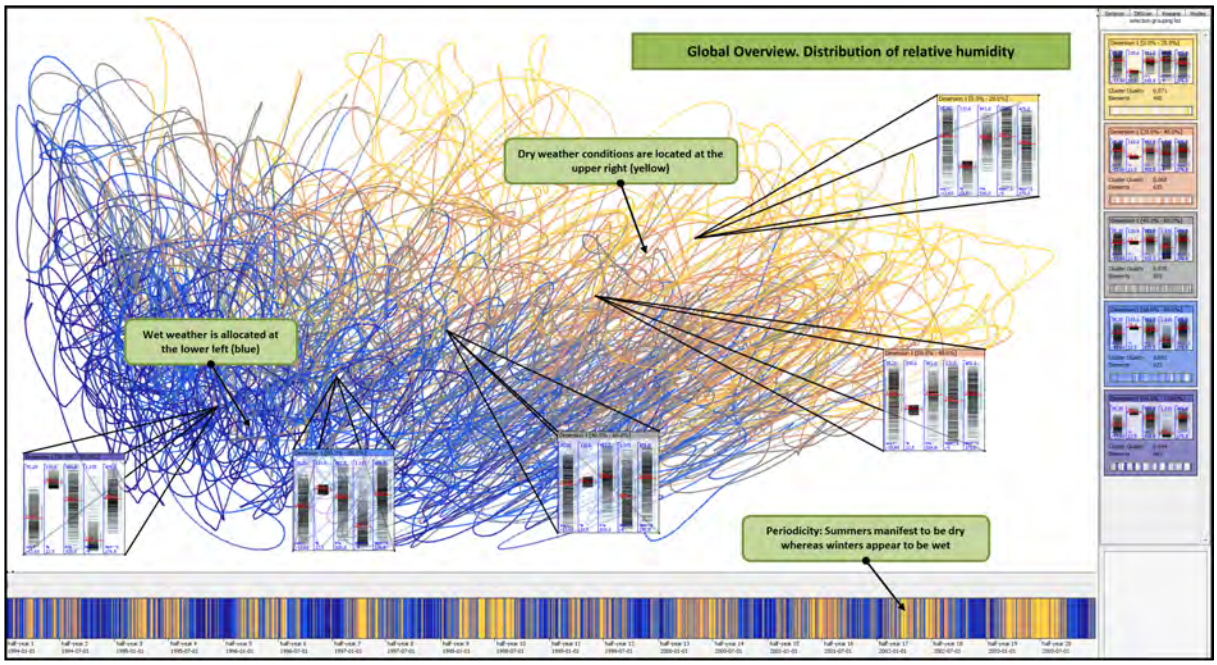


Figure 6: Data aggregation on single input data dimensions: Distribution of relative humidity values (color map: blue means wet, yellow means dry). We constitute rainy weather states to be located left on the Time Series Path Map. By exploring the Color Time Bar, we discover rainy weathers dominating the winter periods. Composing these two findings, we reason that (wet) winter climates are located on the left of the Time Series Path Map.

views. We give a short introduction to the major visual-interactive capabilities of the TimeSeriesPaths system.

Tooltipping

An important user requirement is detail on demand visualization. By hovering above data elements on the Time Series Path Map and the Color Time Bar, tooltips show the multivariate data information and the position of the respective data elements on the time axis (cf. Figures 9 and 10).

Selection, Interactive Grouping and Highlighting

The selection of data is supported in each of our three views. The user can (1) select single data points, (2) time series paths or subsequences thereof, (3) the selection of data within a distinct display region in the Time Series Path Map is possible (cf. Figure 7). The user sketches an arbitrarily polygonal shape on the map, and the surrounded data points will be selected.

Data selections can subsequently be added to the Data Aggregation List for additional information about the selection and for future re-selection. The respective selection is highlighted in all three views to allow the user the detection of interesting patterns. For example, when the user selects a data cluster from the Data Aggregation List (cf. Figures 9 and 8), respective data points are highlighted in the Time Series Path Map and the Color Time Bar. Thus, the user has three different scopes for the exploration of the selected data: (a) the distribution of the data on the Time Series Path Map, (b) occurrences of data elements along the time line in the Color Time Bar and (c) cluster value distributions in the Data Aggregation List (cf. Figure 8).

By means of transparency and plotting size, the user can counter over-plotting on his own by reducing the visibility of elements that are not selected.

Rollercoaster Animation

The Color Time Bar also contains a *Time Slider* for animated time series analysis. We can drag the Time Slider to a specific point or interval in time, and corresponding subsequences are highlighted with circular shapes in real-time on the Time Series Path Map. A schematical demonstration of our so called "Rollercoaster Animation" is given in Figure 4, an application is shown in Figure 10. This interactive animation allows a detailed exploration of the distribution of projected values over time, and also to detect periodic patterns on the Time Series Path Map. The latter is especially helpful in case of over-plotted displays, where a large amount of data elements is visualized on the display.

5 CASE STUDY

We apply our system to a data set from earth observation research. Based on consultation with domain researchers, we explore weather phenomena hidden in the

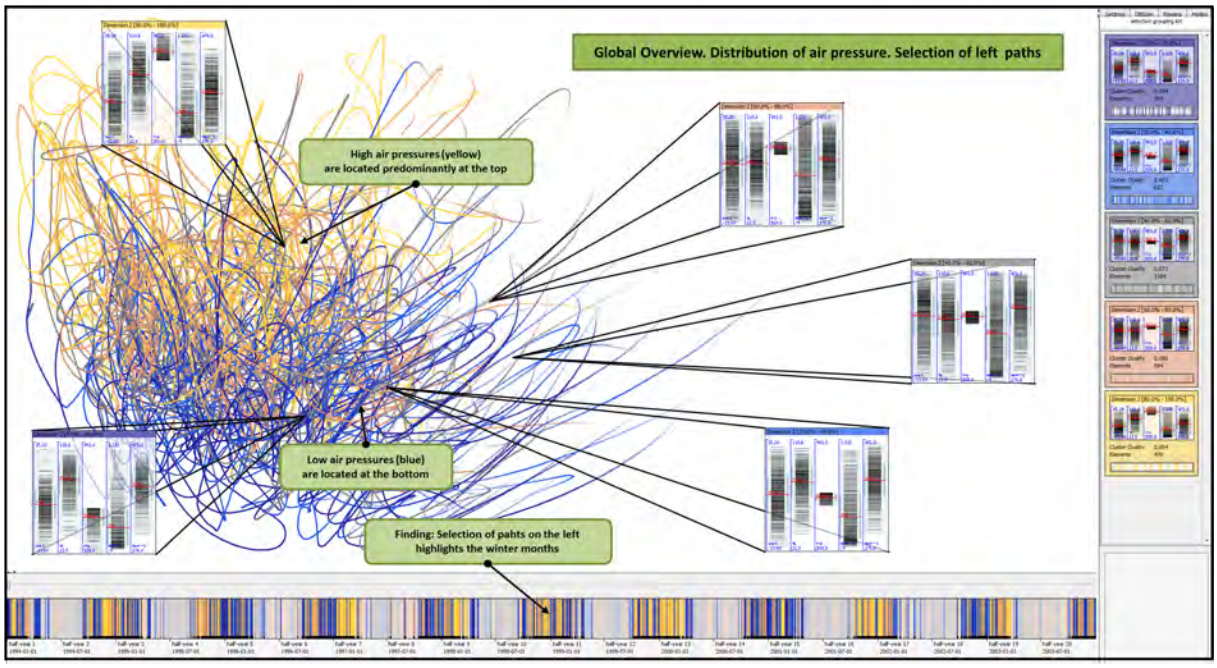


Figure 7: Data aggregation on single input data dimensions: Air pressure development. Selection of the left half of the paths (winter weathers). We discover a color gradient from high (top, yellow) to low (bottom, blue) air pressure values.

data like periodic patterns, frequent 'weather states' and abnormal behavior that can be found with our system.

5.1 Data Set and Application Domain

Our considered data set is acquired from the open data repository PANGAEA [21], operated by the Alfred Wegener Insitute (AWI) for Polar and Marine Research in Bremerhaven. PANGAEA archives and publishes geo-referenced scientific earth observation data in the research areas of water, ice, sediment and atmosphere. Our data set focuses on atmospheric weather measurements, gathered in the scope of the Baseline Surface Radiation Network (BSRN) [20] PANGAEA compartment. These measurements are multivariate atmospheric observations of radiation-related physical parameters) which were recorded every minute. We focus on a dataset of ten years duration, originated from the BSRN station in Payerne (Switzerland) in the time period of January 1st, 1994 to December 31th, 2003 [4]. Payerne is located in the center of the triangle Lyon, Milan and Fribourg at 491 meters above sea level. The climate of Payerne is temperate, semi-continental with average minimum temperatures at about -2°C in January and about 18°C in July. The average daily sunshine duration varies between 2 hours in January and 8 hours in July. Hence, the researchers affirm a yearly climate periodicity to the data that serves as ground truth and primary analysis goal. Beyond that, the so called "summer of the century" in 2003 produced temperature values up to 40°C and motivates us finding this and yet other anomalies in the data set.

We consulted researchers from BSRN to select a suitable parameter subset for detecting interesting weather scenarios. Besides *temperature*, *relative humidity* and *air pressure*, we incorporate the *short-wave downward radiation (SWD)* and the *long-wave downward radiation (LWD)*. The SWD is well suited to give statements about cloud occurrences. Most radiation is measured at the so called clear-sky condition, even when there are no clouds in the sky. It is used for climate research in general and in applied sciences, e.g., in land surface assimilation models, surface energy budget models, and ocean assimilation models. In agriculture, the short-wave downward radiation is used as an input for crop modeling and the solar industry applies it for estimations where to build solar power plants. The LWD is another important factor in the energetic exchange between atmosphere and the earth surface. While the solar dependent short-wave downward radiation is near zero at night, the long-wave downward radiation can be measured all night long. The long-wave downward radiation is higher when the sky is clear. By applying these five measurements as our data set, we are able to make statements about different weather states that possibly change within a seasonal cycle.

Due to the long time period of ten years, we determine each single day as one data point, periodic behaviors within single days are also discovered in the data set and possible to analyze with our system, but not in the focus in this case study. In order to remain on a uni-

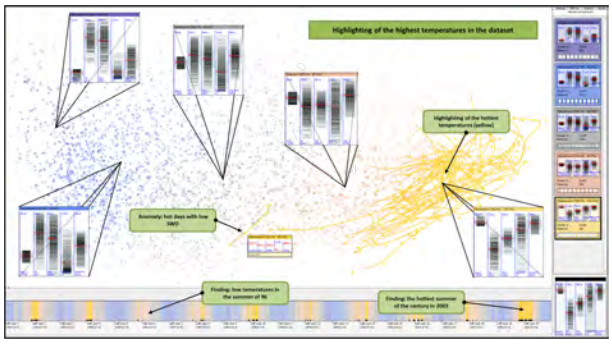


Figure 8: Advanced exploration of Figure 1. We have no problems in identifying the periodic appearance of hot temperatures in the summers in the Color Time Bar. Furthermore, the "summer of the century" anomaly in 2003 stands out with a lot of hot days.

versally accepted approach, we use a generic moving average routine to produce meaningful daily representatives, climate specific daily data aggregation procedures are not applied. Missing value periods of less than 6 hours are linearly interpolated, longer data gaps are ignored. We want to point out that other preprocessing approaches are possible and merely implicate for us the effort of reconfiguring parameters or, if necessary, add a domain-specific preprocessing routine.

5.2 Obtaining a Global Overview

We primarily obtain a global overview to the Time Series Path Map and the data, respectively (cf. Figures 1 and 6). This is crucial due to the described problems in dealing with large multivariate data and projection-based approaches (cf. Section 3). The Color Time Bar indicates a meaningful periodicity with in the seasonal cycle. We constitute Payernes climate to be warm in the summer and cold in the winter period (cf. Figure 1). The overview is completed with Figure 6, where the relative humidity appears to be high (rainy) on the left and low (dry) on the right. At least since the Time Color Bar shows summers to be dry and winters to be wet, we can constitute that the left half of the Time Series Path Map depicts the winter period whereas the summer time is allocated at the right of the display. We prove this hypothesis in Figure 7 by selecting the left half of the time series paths and obtain a meaningful segmentation on the Color Time Bar between summer and winter. Taking the cluster glyphs of the three discussed images into account, we assess correlations between dimension 1 (temperature), 4 (SWD) and 5 (LWD) and thus register another finding in the data set.

After we have received a global overview to the data and our views (some findings may appear evident to the reader so far), we now proceed our case study and focus on the exploration of more particular findings.

5.3 Findings in the Data Set

We now focus on abnormal behavior and anomalies in the data set. We try to discover the "summer of the century" of 2003 as a first finding. We use the view shown in Figure 1 and select the hottest data cluster (yellow); the result is shown in Figure 8. Besides, we discover the coldest summer of the data set in the year 1996 as a new finding. Together with the researchers from AWI, we find our final data exploration goal in the detection of thunderstorms and intense low-pressure systems. Besides the researchers expertise, we consult Internet forums, weather history literature, and insurance damage reports to verify our findings. Figure 9 displays our course of exploration. We focus on the air pressure dimension and apply our *gradient-based* statistical property that measures value changes over time. An aggregation to six clusters produces one group of about 200 highlighted data points that manifest extremely decreasing air pressure gradients. We tooltip a collection of five proven hurricanes and chose the most prominent and devastating hurricane *Lothar* for a detail on demand exploration. Figure 10 details about the air pressure development over 10 days in december 1999. The Rollercoaster Animation helps us navigating through a clearly arranged display, released from visual clutter and overview problems.

6 DISCUSSION

One of the most traditional visualization types for time series data are line charts. In case of multivariate time series, multiple parallel line charts can be used for data visualization. Eventually, projection-techniques such as studied in this paper need to be compared against alternative time series visualization options. While we have not yet done a formal comparison, we provide a conceptual discussion to point at possible advantages and disadvantages of the projection-based approaches vs. line chart approaches.

First, we expect the traditional line chart approach to have scalability problems with respect to very long time series, and to a high number of dimensions. The projection-based approach for the visualization of time series data aims at improving scalability with respect to (1) the time axis (long time series) and (2) the potentially high number of dimensions. Considering (1), information loss occurs for line charts as soon as the number of time stamps becomes larger than the number of available pixels on the x-axis of the line chart display. Basically, three observations can be made:

1. Drawing multiple data points per pixel coordinate leads to visual artifacts and information loss.

2. Downsampling the number of time stamps reduces the amount of information of the visualization.

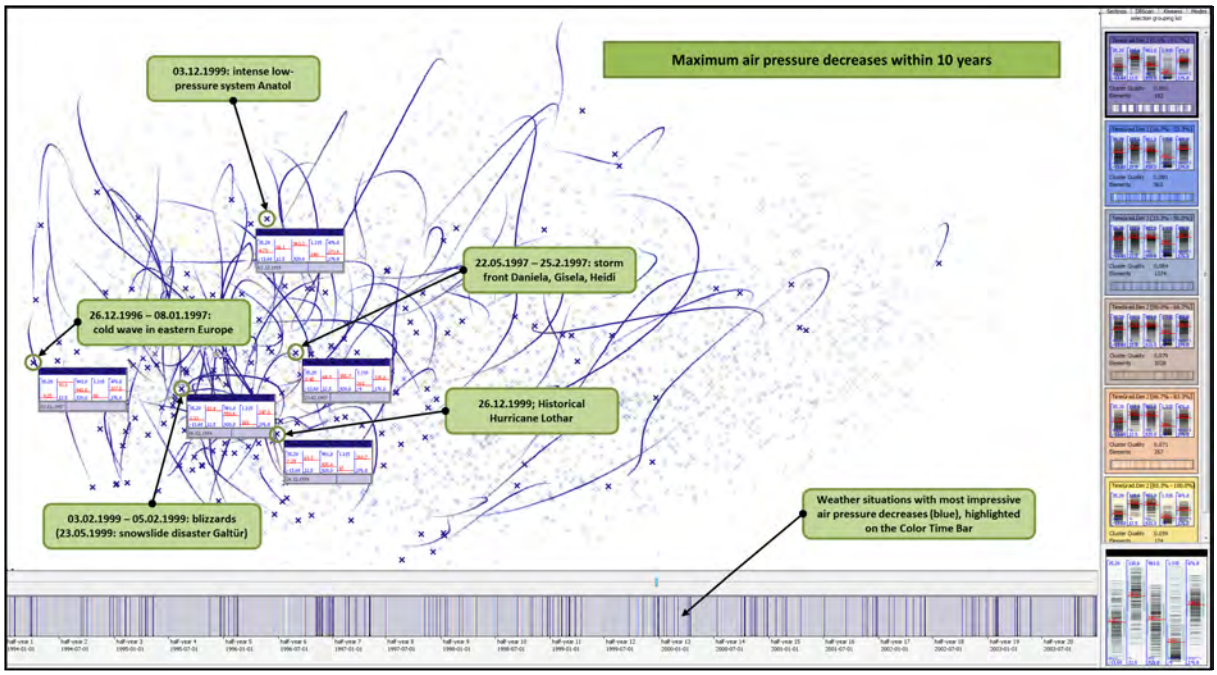


Figure 9: Detection of historic thunderstorms by highlighting most crucial air pressure decreases (blue).

3. Scrolling facilities allow to navigate large time series, yet can be cumbersome for very large time series and may lead to loss of user overview.

Using a 2D projection has the potential to show longer time series in a given display space, if an appropriate projection can be found. On the other hand, interpretation of the projected time series paths may become harder, as directions and distances cannot be read as straightforward as in a line chart.

Considering (2), a high number of dimensions may constitute a problem for multiple line charts. At some point, the available display space is exhausted when too many dimensions are combined in a multiple line chart visualization. In projection, dense data point regions are not only visual clutter. These regions represent dense regions in the input data space and offer potential starting points for further user search and filtering.

The second distinction between multiple line charts and projection concerns the number of data attributes to show. The projection condenses the information of all dimensions in one time series path, providing dimensionality reduction. In general, projection of multivariate data brings up questions about the application-dependent choice of the projection variant (cf. Subsection 3.1) and the preservation of information hidden in the input data. As future work, we need to compare the information preservation of multiple line charts (considering problems for large data or many dimensions) and projection-based time series visualization approaches. One first idea is to define a benchmark data set with periodic behavior that is compared in multiple line charts and in projection-based visualization. At present, we depict that the first two main components of the PCA-based 2D projection approach preserve 78% of the chosen 5D input data information in our weather data case study. Thus, we may assume that the amount of used information is rather high. Yet, more precise evaluation and comparison of the information contents and usage in parallel line charts and in projection-based approaches is needed.

7 CONCLUSION

We presented a system for the analysis of multivariate time-series data. The identification of relations between multiple attributes is an intrinsically difficult problem, even with a viable projection to 2D-space. In order to make such a visualization understandable for domain-experts, our system provides methods for statistical aggregation and clusterings, which can be steered by the user in a very flexible way. Beyond just showing cluster IDs we propose a new glyph-based visualization. This glyph shows the multivariate profiles of the clusters and allows for an effective comparison and interpretation of their properties. The system provides linked views to relate different perspectives of the data to each other. In cooperation with earth observation researchers, we tested the usefulness of the approach with a dataset for atmospheric weather measurements over a ten-years time frame.

We believe that the approach presented in this paper is easily applicable to time-series of different domains. In future projects we will apply and test this system with consumption data of the electric power grid. We used

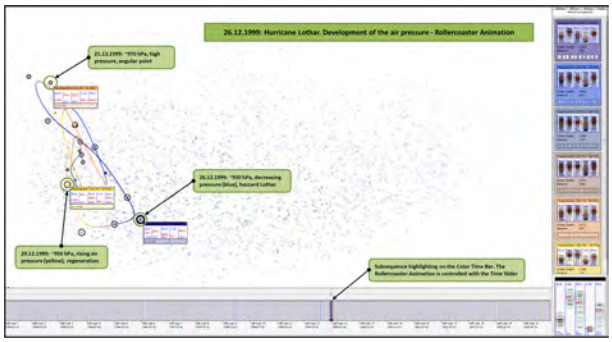


Figure 10: Rollercoaster Animation on hurricane Lothar. Air pressure coloring (blue means low).

projection techniques as an overview because of their popularity as a method for multivariate analysis. However, the methods to calculate, steer and explore the clusters are not restricted to a specific type of overview. In future, we will extend the linked views by other visualizations for multivariate time-series to test for the most effective combination of domain, overview and aggregation methods.

ACKNOWLEDGMENTS

We thank the Alfred Wegener Insitute (AWI) in Bremerhaven, particularly Rainer Sieger, Hannes Grobe and Gert König-Langlo, and everyone involved with PANGAEA for supporting this research effort.

8 REFERENCES

[1] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer-Verlag New York Inc, 2011.

[2] Mihael Ankerst, Daniel A. Keim, and Hans-Peter Kriegel. Recursive pattern: A technique for visualizing very large amounts of data. In *Proceedings of Visualization '95, Atlanta, GA*, pages 279–286, 1995.

[3] Jürgen Bernard, Jan Brase, Dieter W. Fellner, Oliver Koepler, Jörn Kohlhammer, Tobias Ruppert, Tobias Schreck, and Irina Sens. A visual digital library approach for time-oriented scientific primary data. In *Proc. European Conference on Digital Libraries*, pages 352–363, 2010.

[4] Jürgen Bernard, Nils Wilhelm, Maximilian Scherer, Thorsten May, and Tobias Schreck. Reference list of 120 datasets from time series station payerne used for exploratory search. doi:10.1594/pangaea.783598, 2012.

[5] Lior Berry and Tamara Munzner. Binx: Dynamic exploration of time series datasets across aggregation levels. In *Proc. IEEE Symposium on Information Visualization*, 2004.

[6] D. Bhattarai and B.B. Karki. Visualization of atomistic simulation data for spatio-temporal information. In *The 14th Int'l. Conf. on Central Europe in Computer Graphics, Visualization and Computer Vision (WSCG'06)*, 2006.

[7] Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition, 2000.

[8] Tim Dwyer and David R. Gallagher. Visualising changes in fund manager holdings in two and a half-dimensions. *Information Visualization*, 3:227–244, December 2004.

[9] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, Dominik Morent, and Jörn Schneidewind. Intelligent visual analytics queries. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 91–98, 2007.

[10] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. Importance driven visualization layouts for large time-series data. In *Proc. IEEE Symposium on Information Visualization*. IEEE Computer Society, 2005.

[11] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[12] Yueqi Hu, Shuangyuan Wu, Shihong Xia, Jinghua Fu, and Wei Chen 0001. Motion track: Visualizing variations of human motion data. In *PacificVis*, pages 153–160, 2010.

[13] N.K. Kasabov and Q. Song. Denfis: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *Fuzzy Systems, IEEE Transactions on*, 2002.

[14] E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Data Mining, Fifth IEEE International Conference on*, pages 226 – 233. Ieee, 2005.

[15] Teuvo Kohonen. *Self-Organizing Maps*. Springer, Berlin, 3rd edition, 2001.

[16] Zoltan Konyha, Kresimir Matkovic, Denis Gracanin, Mario Jelovic, and Helwig Hauser. Interactive visual analysis of families of function graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1373–1385, November 2006.

[17] J. Lin, E. Keogh, S. Lonardi, J.P. Lankford, and D.M. Nystrom. VizTree: a tool for visually mining and monitoring massive time series databases. In *Proc. of the int. conf. on Very Large Data Bases*, pages 1269–1272. VLDB Endowment, 2004.

[18] Yi Mao, Joshua Dillon, and Guy Lebanon. Sequential document visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13:1208–1215, 2007.

[19] Fabian Mörchen and Alfred Ultsch. Efficient mining of understandable patterns from multivariate interval time series. *Data Min. Knowl. Discov.*, 15(2):181–215, 2007.

[20] A. Ohmura, E. G. Dutton, B. Forgan, C. Fröhlich, H. Gilgen, H. Hegner, A. Heimo, G. König-Langlo, B. mcarthur, G. Müller, R. Philipona, R. Pinker, C. H. Whitlock, K. Dehne, and M. Wild. Baseline surface radiation network (BSRN/WCRP): New precision radiometry for climate research. *Bull. Amer. Met. Soc.*, 79:2115–2136, 1998.

[21] PANGAEA - Data Publisher for Earth and Environmental Science. http://www.pangaea.de/. Last accessed on April 5, 2012.

[22] Y. Sakamoto, S. Kuriyama, and T. Kaneko. Motion map: image-based retrieval and segmentation of motion data. In *Proc. 2004 ACM SIGGRAPH/Eurographics symposium on computer animation*. Eurographics Association, 2004.

[23] Geoffroy Simon, Amaury Lendasse, Marie Cottrell, and Université Paris. Long-term time series forecasting using self-organizing maps: the double vector quantization method, 2003.

[24] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, Cheshire, CT, USA, 1986.

[25] Jarke J. Van Wijk and Edward R. Van Selow. Cluster and calendar based visualization of time series data. In *Proc. IEEE Symposium on Information Visualization*, pages 4–9. IEEE Computer Society, 1999.

[26] Matthew O. Ward and Zhenyu Guo. Visual exploration of time-series data with shape space projections. *Eurographics / IEEE Symposium on Visualization (EuroVis)*, 30(3), 2011.

[27] M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. In *proceedings of the IEEE Symposium on Information Visualization*, pages 7 – 13, 2001.

[28] H. Ziegler, M. Jenny, T. Gruse, and D.A. Keim. Visual market sector analysis for financial time series data. In *Visual Analytics Science and Technology (VAST), IEEE Symposium on*, pages 83–90, 2010.

Design and Analysis of Visualization Techniques for Mobile Robotics Development

Alex Kozlov
The University of Auckland, New Zealand
akoz002@aucklanduni.ac.nz

Bruce A. MacDonald
The University of Auckland, New Zealand
b.macdonald@auckland.ac.nz

Burkhard C. Wünsche
The University of Auckland, New Zealand
burkhard@cs.auckland.ac.nz

ABSTRACT

Simultaneous localization and mapping (SLAM) algorithms are of vital importance in mobile robotics. This paper presents novel Augmented Reality (AR) visualization techniques for SLAM algorithms, with the purpose of assisting algorithm development. We identify important algorithm invariants and parameters and combine research in uncertainty visualization and AR, to develop novel AR visualizations, which offer an effective perceptual and cognitive overlap for the observation of SLAM systems. A usability evaluation compares the new techniques with the state-of-the-art inferred from the SLAM literature. Results indicate that the novel correlation and color-mapping visualization techniques are preferred by users and more effective for algorithm observation. Furthermore the AR view is preferred over the non-AR view, while being at least similarly effective. Since the visualizations are based on general algorithm properties, the results can be transferred to other applications using the same class of algorithms, such as Particle Filters.

Keywords

Algorithm visualisation, augmented reality, robot programming, human-robot interfaces

1 INTRODUCTION

Simultaneous Localization and Mapping (SLAM) [SSC90, LDW91] is a popular and important class of estimation algorithms, addressing the challenge of autonomous map-building for mobile robots. A robot must have a model, or a map, of the physical environment in order to carry out useful navigation tasks. The robot must additionally localize itself within the environment. In SLAM the robot autonomously explores and maps its environment with its sensors while localizing itself at the same time. Despite considerable research, open challenges in SLAM include implementations in unstructured, difficult, and large scale environments [BFMG07], multi-robot SLAM [NTC03] as well as SLAM consistency and convergence [MCDFC07].

SLAM development is made more difficult by its probabilistic nature. In SLAM, neither the robot location nor the environment map are known in advance. However, in order to map the environment the robot location needs to be known with accuracy, and in order to localize the robot the environment map needs to be known with accuracy. Visualizations aid the development and testing of SLAM algorithms by revealing relationships between robot and algorithm states and environmental parameters. Existing SLAM visualization techniques are purely virtual and limited to basic state information, thus lacking *perceptual* and *cognitive* overlap between the robot and the human developer [BEFS01].

Augmented Reality (AR) involves spatially registering virtual objects in real time within a view of a real scene [BR05, Azu97]. AR has been used in robotics to enhance the human-robot interface, but has never been applied to SLAM visualization. This paper presents and evaluates novel AR visualization techniques for SLAM with the purpose of assisting algorithm development and debugging. The introduced concepts and lessons learned are applicable to other estimation algorithms in robotics and related fields.

Section 2 outlines the SLAM algorithms for which the visualizations have been developed. Section 3 presents a review of fields we draw on. Section 4 summarizes the visualization requirements. Section 5 explains the visualization techniques. Section 6 presents the evaluation of the visualizations, and section 7 concludes our paper.

2 SLAM BACKGROUND

The two most popular SLAM algorithm archetypes, the Extended Kalman Filter (EKF) SLAM [GL06, DWB06] and FastSLAM [MTKW03, Mon03], are both based on recursive Bayesian estimation. These

algorithms were targeted for visualization because they are the two most important and prevalent SLAM solution methods [DWB06]. The visualizations presented in this paper would also be relevant for any modified Kalman Filter or Particle Filter based algorithm.

2.1 EKF SLAM

In feature-based EKF SLAM the state is represented by a multivariate Gaussian distribution with mean x and covariance P:

$$x = \begin{bmatrix} x_r \\ x_{f_1} \\ \vdots \\ x_{f_n} \end{bmatrix} \tag{1}$$

$$P = \begin{bmatrix} P_r & P_{r,f_1} & \dots & P_{r,f_n} \\ P_{f_1,r} & P_{f_1} & \dots & P_{f_1,f_n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{f_n,r} & P_{f_n,f_1} & \dots & P_{f_n} \end{bmatrix} \tag{2}$$

x_r is the estimated robot pose and $x_{f_i}, i = 1,\dots,n$ is the estimated position of an environment feature f_i. The main diagonal elements $P_r, P_{f_1}, \dots, P_{f_n}$ are error covariance matrices of the robot pose and the landmark locations. The off-diagonal elements are cross-covariance matrices between the robot and feature positions. The recursive estimation steps of the algorithm are *motion prediction*, *data association*, *observation update* and *feature initialization*.

2.2 FastSLAM

In FastSLAM, which is based on Rao-Blackwellized Particle Filters, the state is represented by a set of N particles:

$$\{w^{(i)}, X_r^{(i)}, \mu_{f_1}^{(i)}, \dots, \mu_{f_n}^{(i)}, \Sigma_{f_1}^{(i)}, \dots, \Sigma_{f_n}^{(i)}\}_i^N \tag{3}$$

where for particle i, $w^{(i)}$ is the particle weight, $X_r^{(i)}$ is the sampled robot path, and each map feature f_j is represented independently by a Gaussian distribution with mean $\mu_{f_j}^{(i)}$ and covariance $\Sigma_{f_j}^{(i)}$. The recursive estimation steps of the algorithm are *motion prediction*, *weight update*, *resampling* and *map update*.

3 LITERATURE REVIEW

A number of *Robotics Development Environments (RDEs)* are available for use in robotics programming, but none offers visualizations for SLAM. Examples include Player [GVS+01], CARMEN (Carnegie Mellon robot navigation toolkit) [MRT] and Pyro (Python Robotics) [BKMY03]. These offer purely virtual sensor data visualizations. Possibly the most flexible support for visualizations is offered by ROS (Robot Operating System) [QGC+09], which includes a variety of data-types such as point clouds, geometric primitives, robot poses and trajectories.

No formal studies have been done for visualization techniques in SLAM. The SLAM visualization state of the art is inferred from the visual representations of SLAM systems and data in the literature. The current "conventional" method of visualizing EKF-style SLAM is by showing the mean estimates for the robot and features, along with the covariance ellipsoids showing the individual uncertainties (e.g. [BNG+07, NCMCT07]). For Particle Filter type SLAM, all current robot poses and mean feature locations are shown for all particles (e.g. [MTKW03, Mon03]). Perhaps the most interesting example of an existing SLAM visualization is the 3D graphical representation of the robot in relation to the mapped obstacles, with the uncertainties shown by dotted lines around the objects [NLTN02]. Martinez-Cantin et al. visualized a constructed 3D map from the robot's point of view [MW03].

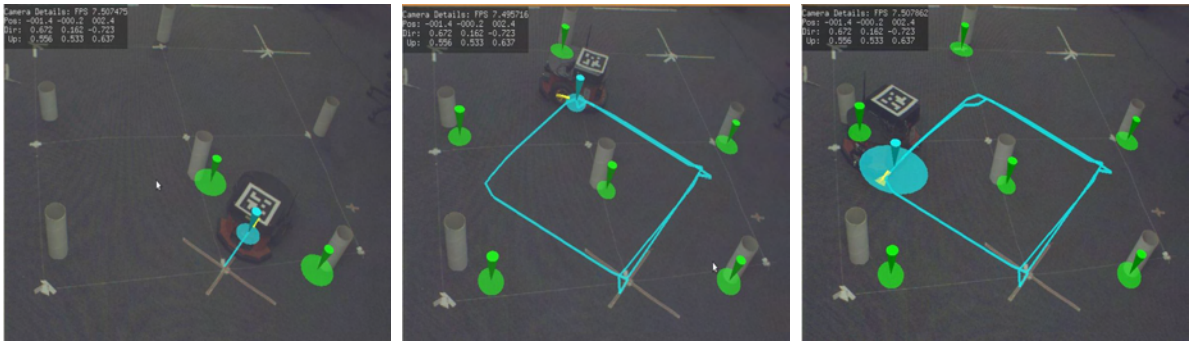
None of the basic SLAM visualizations suggested so far employs an AR environment. However, AR systems have been developed and used in robotics. An example is *ARDev* [CM06, CM09]. It provides perceptual overlap between the human and the robot by visualizing sensor data within the robot's real world environment. Nunez et al. use AR for supervision of semi-autonomous robot navigation tasks [NBPLS06]. A topological map is generated online and visualized with AR. Daily et al. use AR to supervise a swarm of 20 robots for search and rescue scenarios [DCMP03]. Virtual arrows above every swarm member in view convey the intention and direction of travel. AR has also seen considerable application in mobile robot tele-operation [BOGH+03] and manipulator robotics [NCE+07].

4 VISUALIZATION REQUIREMENTS

The underlying requirement for all of the visualizations is to embed information within the context of the real world environment the mobile robot operates in. This provides a qualitative comparison between the estimates and the ground truth, and shows sources of potential errors within the real environment.

4.1 EKF SLAM Requirements

The fundamental EKF SLAM requirement is to visualize the state and the individual uncertainty covariances. The state consists of 2D robot and feature locations, and the robot orientation in the ground plane. The 2 by 2 covariance matrices for the robot and each feature indicate the positional uncertainty, together with the robot orientation variance.



(a) (b) (c)

Figure 1: **Conventional AR EKF Visualization:** EKF-SLAM state and covariance visualization in AR, showing progression over time. The AR view provides a direct comparison of the estimates against the ground truth in the real world environment; this shows the discrepancies between the truth and the estimates.

Correlations between features are well known to be important for correct SLAM operation [DWRN96]. In [DNC+01] feature correlations are defined as follows. Let d_{ij} be the relative position between any two feature estimates f_i and f_j. Let $P_{d_{ij}}$ be the covariance of d_{ij} as follows:

$$d_{ij} = x_{f_i} - x_{f_j} \tag{4}$$

$$P_{d_{ij}} = P_{f_i} + P_{f_j} - P_{f_i,f_j} - P_{f_i,f_j}^T \tag{5}$$

$P_{d_{ij}}$ is a measure of relative error between the two features, and is therefore also a measure of their correlation. The expected convergence property [HD07, DNC+01] is that correlations strengthen as observations are made over time, i.e. the volume of uncertainty in $P_{d_{ij}}$ is non-increasing. Visualization of the correlation behaviour is essential. Violations of this behaviour (i.e. weakening correlations) indicate problems in the SLAM system, and therefore must be detected by the user. Specifically, the 2 by 2 covariance matrix $P_{d_{ij}}$ must be visualized for all feature pairs, together with its non-increasing volume of uncertainty trend. Violations must be exemplified.

4.2 FastSLAM Requirements

The fundamental FastSLAM requirement is to visualize the state represented by the set of particles. This means visualizing 2D points for the robot and Gaussian mean feature locations, for all particles. Additionally robot orientations for all particles must be visualized.

Due to the Rao-Blackwellized structure of FastSLAM, sampling is only done on the robot path. The error in the map estimation for a given particle is dependent on the accuracy of the robot path. For this reason, it is important to visualize the relationship between the robot path and map estimates within particles, or intra-particle associations. Specifically, this refers to visually linking estimates from the same particle, and distinguishing these from other particles. Visualization of the individual weight for each particle is also important in order to gain insight into the resampling phase of the algorithm.

Lastly, a more qualitative and more intuitive representation of the SLAM solution produced by the filter would be useful. This needs to show a better overall picture of the solution, possibly at the cost of lower information content or being less exact.

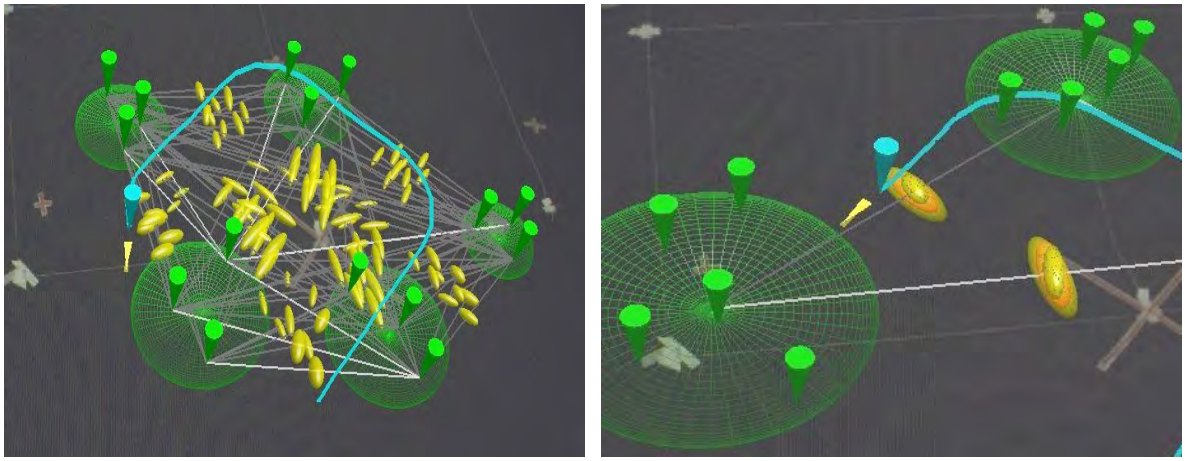
5 AR VISUALIZATION TECHNIQUES

5.1 EKF SLAM Visualizations

5.1.1 Conventional EKF SLAM Visualization

Fig. 1 presents the state-of-the-art conventional EKF visualization implemented in AR. The underlying real world images present an overhead view of the robot and its environment. The robot is a PIONEER 3-DX [Rob08]. The map the robot is building consists of two dimensional points in the ground plane represented by white cardboard cylinders. The cylinders are the only physical objects being mapped and are extracted from raw laser rangefinder data. The robot drives around and performs SLAM in a small 1 by 1 meter loop. The graphical icons augmenting the video frames represent SLAM information:

- **Cyan Marker** - The cyan downward pointed cone represents the estimated robot position. The cyan ellipsoid underneath is the robot position covariance. The cyan line is the robot path.
- **Green Marker** - The green downward pointed cone represents the estimated feature position. The green ellipsoid underneath is the feature position covariance.
- **Yellow Marker** - The yellow triangular pointer represents the robot orientation estimate. A semitransparent yellow circular wedge represents the orientation variance.



(a) (b)

Figure 2: **EKF SLAM Correlations Clustering.** (a) shows all inter-cluster correlations, (b) shows only the maximum, mean, and minimum inter-cluster correlations. Green wireframe circles represent spatial clusters.

The markers represent the SLAM estimates for qualitative real-time visual comparison against the ground truth presented in the real image, i.e. the green markers correspond to the white cardboard cylinders and the blue marker to the physical robot. For the orientation an "arrow" type marker was chosen, as commonly used in SLAM and robotics visualizations. For the covariance, the common tensor ellipsoid glyph was used, which is superior to line or arrow type ellipsoid glyphs. Colour was used to define specifically what the estimate refers to, i.e. robot or features. The design follows the "Natural Scene Paradigm", which is based on humans' ability to immediately perceive complex information in a natural scene [WL01].

5.1.2 Correlations Visualization

In previous work [KMW09] we presented the novel feature correlation visualization shown in Fig. 2a. It satisfies the requirements discussed earlier. For every pair of features $\{f_i, f_j\}$ the visualization contains:

- A line linking the feature estimates f_i and f_j
- A yellow tensor "correlation ellipsoid" for $P_{d_{ij}}$ rendered at the half-way point on the line

As $P_{d_{ij}}$ is a two-dimensional covariance, it produces a 2D tensor ellipsoid. However, the problem of visual cluttering becomes evident when many such ellipsoids grow in size and overlap. It becomes difficult to discern any individual ellipsoids. To mitigate this issue the ellipsoids were inflated to a shaded 3D shape. The second eigenvalue is used for the length of the axis into the third dimension. Giving a 3D shaded volume to the correlation ellipsoids provides better visual distinction to overlapping ellipsoids, however a limitation of this method is that it occludes more of the background world image.

Strengthening correlations show up as decreasing volumes of the correlation ellipsoids. If the volume of the correlation ellipsoid *increases* (i.e. the correlation weakens), this is considered a violation of the expected behaviour. This occurrence is exemplified in the visualization by changing the colour of the ellipsoid to red. Thus, the visualization allows the observation of the expected correlations trend, and the detection of its violations.

The problem of visual cluttering is resolved by *spatial clustering* using the standard single-linkage hierarchical clustering method [LL98]. Features are divided into spatial clusters, and only the correlations between features in different clusters are shown. The green wireframe circles exemplify the clusters computed by the algorithm. This image demonstrates a pure virtual simulation of the SLAM algorithm, and hence no physical robot and environment is shown.

In order to further reduce the number of correlations in view the user can select to only see the minimum (yellow), mean (orange), and maximum (yellow) inter-cluster correlations (Fig 2b). The expected correlation convergence can be observed through the non-increasing size of the minimum correlation ellipsoid [Koz11].

5.2 FastSLAM Visualizations

5.2.1 Conventional FastSLAM Visualization

Fig. 3 presents the conventional state-of-the-art FastSLAM visualization implemented for the first time in AR. The underlying real world images present an overhead view of the robot and the environment the robot is working in. The graphical icons augmenting the video frames represent SLAM information:

(a) (b) (c)

Figure 3: **Conventional FastSLAM AR Visualization:** particle representation for the robot pose and features, showing the joint SLAM posterior distribution computed by the particle filter and its progression over time.

- **Cyan Marker** - The cyan downward pointed cone represents the sampled robot location for a given particle.
- **Yellow Marker** - The yellow arrow-type marker represents the sampled robot orientation for a given particle.
- **Green Marker** - The green downward pointed cone represents a Gaussian mean feature location for a given particle.

The visualization shows the joint SLAM posterior probability distribution of the robot pose and the feature map. As for the EKF, the markers represent the SLAM state for qualitative visual comparison against the ground truth presented in the real image, i.e. the green markers correspond to the white cardboard cylinders and the blue marker to the physical robot.

5.2.2 *Colour Mapping Visualizations*

Fig. 4 presents a colour-mapping visualization technique addressing the requirements of intra-particle associations and particle weights, as discussed earlier. First the centroid and maximum distance from the centroid are computed for the current robot positions in the particles. This creates a local polar coordinate frame for the robot positions (and thus the particles they belong to), originating at the centroid. Then each particle's polar coordinates are mapped directly onto the Hue and Saturation parameters in the HSV colour model. Thus, each particle which has a unique robot position is assigned a unique colour. This colour is used to encode members of the same particle (intra-particle associations), e.g. a red feature and a red robot pose belong to the same particle. This shows the important relationship between the map and robot path estimations. In the final step, the particle weight is encoded into the Value (brightness) parameter of the HSV model. Lighter coloured markers indicate higher weights, and darker colours indicate lower weights. Fig. 5 shows the

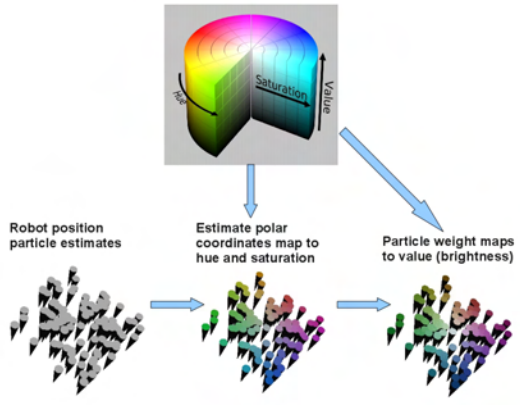


Figure 4: **The colour-mapping technique.**

colour-mapping visualization applied in SLAM. The colour-coded relationship between the robot position and feature errors is clearly visible.

5.2.3 *Surface Density Visualization*

Fig. 6 presents a novel surface density visualization technique developed for FastSLAM. The purpose of this visualization is to present a better overall qualitative picture of the SLAM solution produced by the filter. Here the joint SLAM posterior probability of the robot and the features is represented by a smooth, shaded 3D surface. The mapping area is divided into an uniform (customizable) grid, where the density of a given cell is given by the number of particle members (robot pose or features) within that cell. Then the surface is interpolated over the grid using a *Radial Basis Function (RBF)* [Buh03], with the density giving the surface height. If colour-mapping is enabled, the colour for each cell is the average of the particle colours within it. Otherwise, the cyan surface represents the robot pose and the green surfaces the features. In addition, a single arrow is drawn above each cell of the robot pose surface. This is the average robot orientation within the cell.

(a) (b) (c)

Figure 5: **Intra-particle Associations Visualization:** colour-mapping used to show members belonging to the same given particle, showing progression over time. Brightness indicates particle weights.

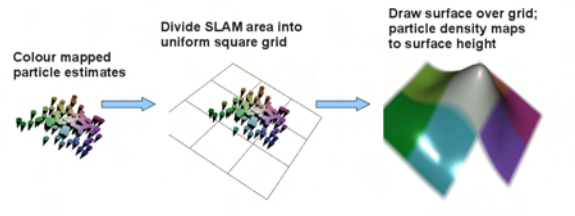


Figure 6: **The surface colour-mapping technique.**

Fig. 7 shows the surface density visualization without the colour mapping. Intuitively the height of the surface indicates the SLAM posterior probability. The shape of the surface provides a good qualitative picture of the uncertainty spread of the distribution, as compared to rendering each individual marker. Fig. 8 shows the colour-mapped surface density visualization. This offers the benefits of both the surface and the colour-mapping techniques. The visualization shows both the shape of the uncertainty spread and the colour-mapped intra-particle associations.

6 EVALUATION

6.1 Experimental Setup

The visualization system was implemented with a ceiling mounted Prosilica EC 1350C Firewire camera for image capture. Registration was done using ARToolKitPlus and a fiducial marker mounted on the robot. The robot's initial position is registered in the AR coordinate frame as the origin of the SLAM map. This allows the registration of the SLAM data in the AR coordinate frame. Videos were taken of the robots SLAM performance using different visualization techniques for correctly implemented SLAM algorithms and versions with typical errors we inferred from the literature and a previous user survey [Koz11].

6.2 Methodology

We performed five experiments summarised in table 1 in order to investigate the effectiveness of the visualizations for assisting SLAM development. In particular we evaluated AR-based visualisation techniques versus non-AR visualisation techniques and novel AR visualisation versus AR-implementations of techniques considered current state-of-the-art. The purpose of the study was to compare the effectiveness of the visualization techniques for SLAM algorithm development, i.e. fault detection and fault correction.

Fault Detection Experiments		
Experiment	**Vis 1**	**Vis 2**
EKF Exp 1	Conventional AR	Conventional non-AR
EKF Exp 2	Correlations AR	Conventional AR
FastSLAM Exp 1	Conventional AR	Conventional non-AR
FastSLAM Exp 2	Colour-mapping AR	Conventional AR
FastSLAM Exp 3	Surface density AR	Conventional AR

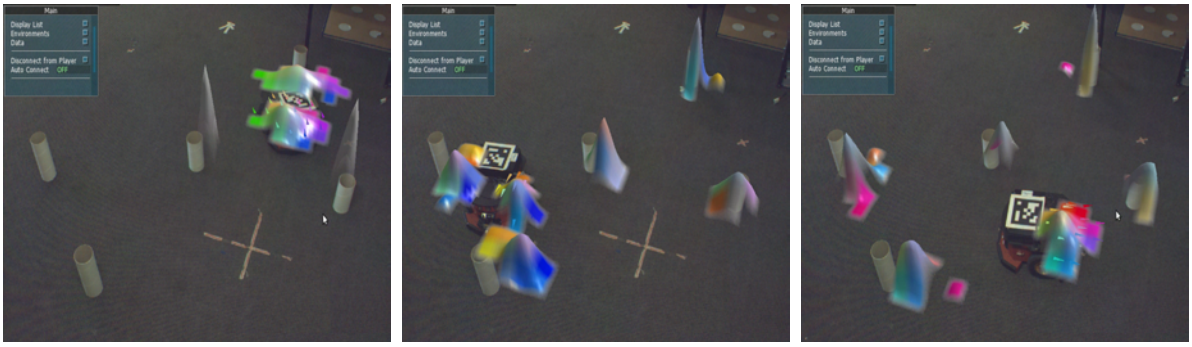
Table 1: Fault detection experiments summary. Each experiment compared Vis 1 with Vis 2.

The experiments were performed as a web-based survey questionnaire. Participants were invited over email, through the major international robotics mailing lists. These included Robotics Worldwide, Australian Robotics and Automation Association (ARAA) and European Robotics Research Network (EURON). Ideally the desired population of the participants would be SLAM developers; but in practice to obtain sufficient participants the population scope was widened to *robotics* developers. The experiments involved participants watching online videos of the visualizations and answering questions about the visualizations.

Within the questionnaire document, the concepts of SLAM and AR were first explained, along with in-

(a) (b) (c)

Figure 7: **Surface Density Visualization:** the surface density visualization without the colour-mapping, showing progression over time. The shape and height of the surface conveys the joint SLAM posterior distribution computed by the particle filter.



(a) (b) (c)

Figure 8: **Colour Mapped Surface Visualization:** the surface density visualization with the colour-mapping, showing progression over time. The visualization shows both the shape of the joint SLAM posterior distribution and the associations within particles.

troductory videos and explanations about the visualizations. Each AR visualization was presented with a video of it being used for SLAM with a real robot and cylindrical point features, along with a written explanation. To present the non-AR visualization, two videos were used. One was the virtual SLAM visualization, and the other was the video of the physical robot performing SLAM corresponding to the SLAM visualization.

After showing correct operation, artificial faults were introduced into the SLAM systems. Within each experiment the same fault was used to compare the visualizations, however the visualizations showed the fault in different ways. For each visualization, the participants were asked as a multi choice question what SLAM fault is present in the visualization (if any). For each pair of visualizations compared, the participants were also asked in a short answer question which visualization they felt was more effective (Vis 1, Vis 2, or neither) and why. Details of the study are found in [Koz11].

6.3 Results

There were 24 participants in the EKF evaluation, and 14 participants in the FastSLAM evaluation.

In EKF Exp 1 users detected 75% of errors with the AR visualization and 70% of errors with the non-AR visualization. Users liked that the AR visualization combined a view of the real environment with the SLAM information. Reasons for prefering non-AR were perception difficulties in AR due to the 3D camera perspective, deformation, depth and projection, and the real-world camera image. In FastSLAM Exp 1 all of the participants preferred the AR visualization. In terms of effectivness both visualisations resulted in 57% of errors being detected.

In EKF Exp 2 our new correlation visualization allowed users to detect 79% of errors, whereas the traditional visualization only allowed detection of 50% of errors. Users liked in the correlation visualization the explicit representation of correlation faults enabling a faster detection. Reasons for prefering the conventional AR visualization were clearer, more intuitive representation of robot pose/landmarks and faults therein, the correlation ellipsoids being hard to understand and occluding the landmark/robot markers, and robot/landmark covariances being more representative of the estimation process.

In FastSLAM Exp 2 users were able to detect 64% of errors using colour mapped particles and 35% of errors using the conventional visualization. Users liked about colour-mapping the clear representation of particle weighting and the resampling process, and that colour mapping offers more information in a compact representation allowing for better fault detection.

In FastSLAM Exp 3 users identified 42% of errors using the surface density visualization and 71% of errors using the conventional visualization. Users liked the compact and effective representation of the particle set distribution in the surface density AR visualization, and that the peak of the surface indicates the most likely estimate position whereas the spread shows the amount of divergence in the estimates. However, users complained that the surface representation is too opaque and obscures the true landmarks, and that the surface view does not show the robot orientation clearly. Users stated that the conventional AR visualization is easier to analyze in order to detect errors.

7 CONCLUSIONS

This paper presented novel AR visualization techniques for SLAM algorithms. The visualization requirements were challenging because SLAM algorithms are detailed, complex, and must address real world uncertainties. To address the requirements, visualizations were developed in AR to target the most important aspects of the SLAM algorithms, including feature correlations, particle weights and relationships.

Our Evaluation shows that AR visualizations are preferred over non-AR visualizations, and that the novel techniques for feature correlations is more effective than the existing state of the art for SLAM fault detection. The visualizations are effective because they target specific aspects of the algorithm and because AR enables visualization of the real world and associated uncertainties. The correlation visualization can be adapted to any application requiring representation of correlations between physical entities. Care must be taken that visualization icons do not obscure relevant real-world information in the camera view and that visual complexity does not put undue stress on the user. Hence small point based icons are preferable over more complex and information rich surface representations. The presented visualizations perform differently well for different types of errors. Ideally the user should be able to swap interactively between all of the presented techniques.

8 REFERENCES

[Azu97] R.T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[BEFS01] C. Breazeal, A. Edsinger, P. Fitzpatrick, and B. Scassellati. Active vision for sociable robots. *IEEE Trans. Syst., Man, Cybern.*, 31(5):443–453, 2001.

[BFMG07] J.L. Blanco, J.A. Fernandez-Madrigal, and J. Gonzalez. A New Approach for Large-Scale Localization and Mapping: Hybrid Metric-Topological SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2061–2067, 2007.

[BKMY03] D. Blank, D. Kumar, L. Meeden, and H. Yanco. Pyro: A python-based versatile programming environment for teaching robotics. *Journal on Educational Resources in Computing (JERIC)*, 3(4):1–15, 2003.

[BNG+07] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot. Consistency of the EKF-SLAM algorithm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3562–3568. IEEE, 2007.

[BOGH+03] V. Brujic-Okretic, J.Y. Guillemaut, L.J. Hitchin, M. Michielen, and G.A. Parker. Remote vehicle manoeuvring using augmented reality. In *International Conference on Visual Information Engineering*, pages 186–189, 2003.

[BR05] O. Bimber and R. Raskar. *Spatial Augmented Reality : Merging Real and Virtual Worlds*. A K Peters, Limited, 2005.

[Buh03] M. Buhmann. *Radial basis functions theory and implementations*. Cambridge University Press, 2003.

[CM06] T.H.J. Collett and B.A. MacDonald. Developer oriented visualisation of a robot program. In *ACM SIGCHI/SIGART Human-Robot Interaction*, pages 49–56, 2006.

[CM09] T. H. J. Collett and B. A. MacDonald. An augmented reality debugging system for mobile robot software engineers. *Journal of Software Engineering for Robotics*, 1(1):1–15, 2009.

[DCMP03] M. Daily, Y. Cho, K. Martin, and D. Payton. World embedded interfaces for human-robot interaction. In *Annual Hawaii International Conference on System Sciences*, pages 6–12, 2003.

[DNC+01] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization

and map building (slam) problem. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 17(3):229–241, 2001.

[DWB06] H. Durrant-Whyte and T. Bailey. Simultaneous localisation and mapping: Part 1. *IEEE Robotics and Automation Magazine*, 13(2):99–108, 2006.

[DWRN96] H. Durrant-Whyte, D. Rye, and E. Nebot. Localisation of automatic guided vehicles. In *Robotics Research: The 7th International Symposium (ISRR'95)*, pages 613—-625, 1996.

[GL06] S. Ge and F. Lewis. *Autonomous mobile robots : sensing, control, decision- making, and applications*. Boca Raton, FL CRC/Taylor and Francis, 2006.

[GVS+01] B.P. Gerkey, R.T. Vaughan, K. Stoy, A. Howard, G.S. Sukhatme, and M.J. Mataric. Most valuable player: a robot device server for distributed control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1226–1231, 2001.

[HD07] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *Robotics, IEEE Transactions on*, 23(5):1036–1049, 2007.

[KMW09] Alex Kozlov, Bruce MacDonald, and Burkhard C. Wünsche. Covariance Visualisations for Simultaneous Localisation and Mapping. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2009)*, pages 1 – 10, December 2009. `http://www.cs.auckland.ac.nz/~burkhard/Publications/ACRA2009_KozlovMacDonaldWuensche.pdf`.

[Koz11] Alex Kozlov. *Augmented Reality Technologies for the Visualisation of SLAM Systems*. PhD thesis, The University of Auckland, 2011.

[LDW91] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localisation for an autonomous mobile robot. In *Proc. IEEE Int. Workshop Intell. Robots Syst. (IROS)*, pages 1442–1447, 1991.

[LL98] Pierre Legendre and Louis Legendre. *Numerical ecology*. New York : Elsevier, 1998.

[MCDFC07] R. Martinez-Cantin, N. De Freitas, and J.A. Castellanos. Analysis of particle methods for simultaneous robot localization and mapping and a new algorithm: Marginal-slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2415–2420, 2007.

[Mon03] M. Montemerlo. *FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association*. PhD thesis, Carnegie Mellon University, 2003.

[MRT] M. Montemerlo, N. Roy, and S. Thrun. *Carmen, Robot Navigation Toolkit*. Retrieved June 2007 from `http://carmen.sourceforge.net/`.

[MTKW03] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fast-slam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1151–1156, 2003.

[MW03] I. Mahon and S. Williams. Three-dimensional robotic mapping. In *Australasian Conference on Robotics and Automation (ACRA)*, 2003.

[NBPLS06] R. Nunez, J.R. Bandera, J.M. Perez-Lorenzo, and F. Sandoval. A human-robot interaction system for navigation supervision based on augmented reality. In *IEEE Mediterranean Electrotechnical Conference*, pages 441–444, 2006.

[NCE+07] A. Nawab, K. Chintamani, D. Ellis, G. Auner, and A. Pandya. Joystick mapped Augmented Reality Cues for End-Effector controlled Tele-operated Robots. In *IEEE Virtual Reality Conference*, pages 263–266, 2007.

[NCMCT07] J. Neira, J.A. Castellanos, R. Martinez-Cantin, and J.D. Tardos. Robocentric map joining: Improving the consistency of EKF-SLAM. *Robotics and Autonomous Systems*, 55(1):21–29, 2007.

[NLTN02] P. Newman, J. Leonard, J.D. Tardos, and J. Neira. Explore and return: experimental validation of real-time concurrent mapping and localization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1802–1809, 2002.

[NTC03] J. Neira, J.D. Tardos, and J.A. Castellanos. Linear time vehicle relocation in SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 427–433, 2003.

[QGC+09] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[Rob08] ActivMedia Robotics. *PIONEER P3-DX*. Retrieved January 2008 from `http://www.activrobots.com/ROBOTS/p2dx.html`, 2008.

[SSC90] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

[WL01] Burkhard C. Wünsche and Richard Lobb. A scientific visualization schema incorporating perceptual concepts. In *Proceedings of IVCNZ '01*, pages 31–36, 2001. `http://www.cs.auckland.ac.nz/~burkhard/Publications/IVCNZ01_WuenscheLobb.pdf`.

An Applied Approach for Real-Time Level-of-Detail Woven Fabric Rendering

Wallace Yuen
The University of Auckland,
New Zealand
wyue013@aucklanduni.ac.nz

Burkhard C. Wünsche
The University of Auckland,
New Zealand
burkhard@cs.auckland.ac.nz

Nathan Holmberg
77-Pieces Ltd,
New Zealand
nathan@77-pieces.com

ABSTRACT

Photorealistic rendering of fabric is essential in many applications ranging from movie special effects to e-commerce and fashion design. Existing techniques usually render the fabric's microscale structure. However, this can result in severe aliasing and is unsuitable for interactive cloth simulation and manipulation. In this paper we describe a novel real-time level-of-detail fabric rendering technique. The algorithm adjusts geometry and texture details with changing viewpoint by using a mipmapping approach, in order to obtain a perceptually consistent representation on the screen. Compared to previous work we also introduce more parameters allowing the simulation of a wider range of fabrics. Our evaluation demonstrates that the presented approach results in realistic renderings, increases the shader's run-time speed, and reduces aliasing artifacts by hiding the underlying yarn geometry.

Keywords: fabric rendering, anisotropic materials, real-time rendering, cloth simulation, anti-aliasing, level-of-detail methods

1 INTRODUCTION

Realistic fabric rendering addresses many different areas and industries in computer games and fashion applications. It is a challenging research field due to the complexity of the underlying fabric structure, textures, and materials, which results in complex light interactions and aliasing problems when using a raster representation. Fabric structures vary depending on the manufacturing process, such as weaving and knitting, and the desired fiber properties. Previous research in this field has explored different aspects of this problem, such as rendering complex weaving and knitting patterns, and developing specialized lighting models that simulate light interaction with the yarn geometry and its microstructure.

The modeling of complex weaving patterns and yarn geometry can result in aliasing when the screen resolution is lower than the perceived color variations on the material (caused by the geometry, lighting and texture). This is particularly problematic when animating the fabric using cloth simulations, which creates conspicuous temporal aliasing artifacts. In recent years, many hardware anti-aliasing techniques have been developed for real-time applications such as computer games, but are mainly used as a post-processing remedy. In this paper, we describe a level-of-detail fabric rendering technique for reducing the aliasing artifacts with minimal impact on computation time. This method can be used in conjunction with post-processing anti-aliasing techniques to further reduce the aliasing artifacts.

We want to create a parameterized fabric shaders for fashion design and e-commerce applications. This fundamentally requires fast interactive speed, high memory efficiency, and high robustness to support for a wide range of woven fabrics. We found that the model proposed by Kang [Kan10] would be the most suitable for our needs with several extensions and modifications to the original algorithm [YW11]. We adopted this model and implemented it in OpenGL Shading Language to support real-time parameterization of weaving pattern and yarn geometry.

Section 2 reviews existing fabric rendering techniques. Section 3 introduces previously presented techniques for modeling light interaction and rendering fabric, which form the foundation of our fabric rendering framework. Section 4 proposes our level-of-detail algorithm and improvements to the existing algorithm for real-time fabric rendering. Section 5 presents an evaluation of our framework and Section 6 draws conclusions and suggests directions for future work.

2 LITERATURE REVIEW

Fabric rendering techniques have been an active area of research since the early 1990s. We classify existing techniques into two categories, example-based and

procedural-based models. Example-based models focus on capturing reflectance information of specific material and use the captured data for rendering virtual fabrics. Procedural-based models are empirical mathematical models that use various parameters to control the appearance of fabrics.

2.1 Example-Based Models

Example-based cloth rendering techniques require the capturing of reflectance information of materials. This usually requires modification of the lightings, sensors, and planar examples of the corresponding materials. The reflectance properties of a material for different viewpoints and light directions can be encoded in a Bidirectional Reflectance Distribution Function (BRDF), which is often obtained with a gonioreflectometer [War92].

Daubert et al. [DLH01] proposed a Spatially-Varying Bi-directional Reflection Distribution Function (SVBRDF) specifically for cloth rendering using the Lafortune reflection model. It is designed to render clothes with repeating patterns such as knitting and weaving patterns. A texture map is used as a look-up table, for storing precomputed parameters of the Lafortune reflection model. This method works for both knitted and woven clothes by modeling the structure explicitly through the generation of new triangle meshes, but the computation consists of several rendering passes. Even though many methods have been proposed to obtain a SVBRDF using only a single view of a material sample [WZT+08], SVBRDF is generally very memory intensive, which makes it impractical for real-time applications where material parameters are interactively changed.

Another popular example-based model is the Bidirectional Texture Function (BTF), which captures a material's light interaction for varying light source and camera positions. The BTF captures more effects than the SVBRDF including self-shadowing, occlusion, and inter-reflection effects, and is used for many surface reflectance data measurements. The actual textures of the cloth samples are used and stored as texture maps, and they are used at render time with different parameters such as the illumination, camera position, and the texture coordinates of the object. Due to the higher number of parameters, the BTF suffers from high memory requirements and acquisition costs. Kautz [Kau05] introduced a compression method for the BTFs. He acquires a lower number of images and interpolates between them. Despite these compression approaches, example-based methods still require an over-abundant storage capacity for practical use, and they do not offer enough flexibility for rendering different types of clothes with different weaving patterns.

A volumetric approach that uses the microflake model was proposed by Zhao et al. [ZJMB11]. The approach acquires a volume model of the material that needs to be rendered using a X-ray computed tomography (CT) scanner. The volumetric data acquired is post-processed for orientation extraction and noise removal, and is matched to a photograph of the same material captured to obtain the optical properties for fabric rendering [ZJMB11]. This approach suffers from high memory requirements, due to the size of volumetric data, where each fabric sample takes approximately 7.26GB [ZJMB11]. It is also difficult to acquire equipments for this approach, due to the cost of CT scanners, thus making it difficult to capture different fabrics.

2.2 Procedural-Based Models

Procedural-based cloth rendering techniques are models that are designed based on the analysis of fabric structure. Yasuda et al. [YYTI92] developed shading models for woven cloth by analyzing fiber properties and weaving patterns. The authors proposed a tiny facet model for fabric materials taking into consideration reflection and refraction of multiple fabric layers. Using a multiple layer model, they simulated the scattering effects of fabrics by calculating the light refraction at different layers [YYTI92]. The reflection model assumes a simple percentage of warp and weft yarns in woven clothes and used a non yarn-based reflection. The light interaction with a small area of fabric is calculated by obtaining the total reflections [YYTI92]. Hence, this approach does not explicitly model the weaving patterns, but simulates the appearance of the fabric at a higher level where the weaving patterns are not visible.

Ashikhmin et al. [AS00] developed a microfacet-based anisotropic model that can be used for general materials, and was tested by simulating satin and velvet. The authors take into account the weaving pattern of satin and velvet. For example, satin is modeled by weighting the BRDF values of weft and warp yarns [AS00]. Due to a lack of self-shadowing and light interaction at the yarn-level, this microfacet anisotropic model is too generic to be used directly for fabric rendering, but it formed the foundation for many subsequently developed techniques.

Adabala et al. [AMTF03] use a weaving pattern input defined by the user, and generate a corresponding Anisotropic BRDF, texture, and horizon map for the clothing material. The authors render the fabric based on the weaving pattern input provided by the user to generate the overall appearance of the cloth [AMTF03]. This approach extends previous fabric models and allows more complicated weaving patterns to be defined. However, the authors applied the Ashikhmin-Shirley BRDF [AS00] on the object-level rather than the yarn-level, thus the modeling of light interaction with the fabric lacks realism compared to techniques which calculate light interaction based on yarn material and weaving patterns.

Kang [Kan10] proposed a procedural method that models the reflectance properties of woven fabric using alternating anisotropy and deformed microfacet distribution function. The proposed method is based on the microfacet distribution function (MDF) along with the Ashikhmin-Shirley [AS00] anisotropic shading model. Each sample point on the cloth is classified as a weft or warp yarn, and a corresponding distribution function is used accordingly to calculate the reflectance of that point [Kan10]. The alternating anisotropy approach allows the lighting to be defined for weft and warp thread by rotating the microfacet distribution function. Further deformation of the MDF enables the elaboration of yarn geometries and twisted appearances on the surface of each yarn. This approach enables not only the rendering of anisotropic clothes, but also the rendering of bumpy surfaces created by the weaving structure of the fabrics [Kan10].

Irawan [Ira08] developed a reflectance model and a texture model for rendering fabric suitable for distant and close-up views. The reflectance model is defined by the scattering effect of the fabric, while the texture model incorporates highlights calculated from the reflectance model. The texture model (BTF) is generated on the fly using parameters to control the types of yarn (i.e. staple or filament), and the appropriate weave pattern, and the yarn geometry is captured by using the fiber twisting angle to render the highlight on each yarn. A downside of this approach is the lack of shadowing and the masking effects to render some types of fabrics realistically, such as denim. However, the results look convincing and the approach is fully procedural, with intuitive variables at the fiber level, the yarn level, and the weaving pattern level.

3 FABRIC RENDERING MODEL

This section explains two techniques adopted by us in more detail: Kang's fabric rendering model [Kan10] and the Ashikhmin-Shirley anisotropic shading model [AS00] for capturing anisotropic reflections.

3.1 Ashikhmin-Shirley BRDF

The Ashikhmin-Shirley BRDF [AS00] is given by the following equation:

$$\rho(k_1,k_2) = \frac{p(h)P(k_1,k_2,h)F(k_1h)}{4(k_1n)(k_2n)(nh)} \qquad (1)$$

This equation represents the illumination of a point with the incoming light vector k_1 and outgoing light vector k_2 where additional functions explained below describe the effect of the microfacet structure. The vector n represents the surface normal at a point, and the vector h describes the half vector obtained from the incoming and outgoing light vector. The function $P(k_1,k_2,h)F((kh))$ captures the shadowing effects caused by microfacets. The function $F(kh)$ is the Fresnel reflectance that describes the amount of incoming light that is reflected off the surface specularly. The function $p(h)$ is the MDF given by Equation 2. It describes the illumination effects of weaving patterns in Kang's model [Kan10].

3.2 Microfacet Distribution Function

The microfacet distribution function characterizes a surface's distribution of microfacets, by encoding their normal direction relative to the underlying surface.

$$p(h) = \frac{\sqrt{(x+1)(y+1)}}{2\pi}(h\cdot n)^{xcos^2\phi+ysin^2\phi} \qquad (2)$$

The microfacet distribution function in Equation 2 is used to generate the BRDF. The function captures the visual effects of microgeometry, where the reflected specular light on the surface is proportional to the probability of the microfacet surface normals that are aligned to the half vector. Variables x and y controls the shape of the specular highlight and the intensity in anisotropic reflection.

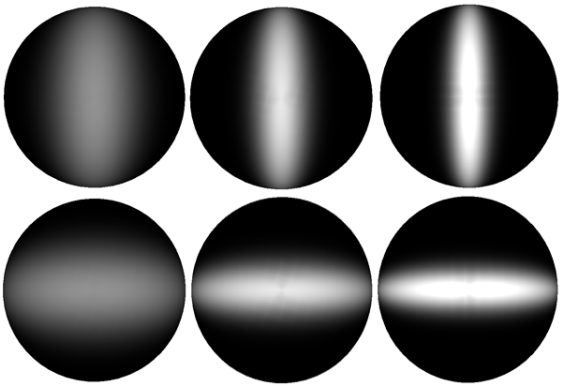


Figure 1: Our generated images using the Ashikhmin-Shirley BRDF [AS00] for visualizing the difference in specular highlights with varying parameters x and y. Top row: x = 10, 30, 50, while y stays constant equal to 1. Bottom row: y = 10, 30, 50, while x stays constant equal to 1.

A visualization of the microfacet distribution is shown in Figure 1. When the x and y values are close to each other, then the distribution of microfacets aligning to h is spread more evenly across the surface. The top row of the diagram demonstrates that if the x-value in the MDF increases from 10 to 50, then the distribution of microfacets becomes denser in the center, thus resulting in a less spread specular lobe on the object surface. This results in an increasingly narrow highlight stretched in y-direction.

3.3 Weaving Pattern Rendering

Kang [Kan10] proposed an alternating anisotropy solution to render weaving patterns by using Equation 2

to generate the specular highlight that can be seen on weaving pattern [Kan10]. Using the fact that weaving patterns are only made of weft and warp yarns, the specular highlight of weft yarns must therefore be obtained by rotating the microfacet distribution by 90°. This is again shown in Figure 1. The rotated microfacet distribution is seen on the second row, and is done by exchanging the values of x and y to create such rotation.

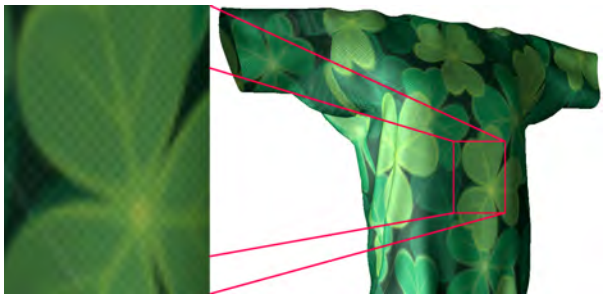


Figure 2: Weave pattern generated by using the alternating anisotropy method showing the closer view (left) and distant view (right).

An example of a weaving pattern generated using the alternating anisotropy method is shown in Figure 2. Without the yarn geometry the weaving pattern looks like a checkerboard pattern, thus the yarn geometry has to be defined for close-up viewpoints.

3.4 Yarn Geometry

The yarn geometry is generated after the weaving pattern by normal perturbation at run-time. Kang [Kan10] proposed that the yarn geometry can be specified by several parameters including: number of threads for each strand of yarn N_ξ, fiber curvature of each strand c_f, and the yarn curvature c_y. Therefore, the normal of each point is altered using these parameters and its sampling position on the weft or warp yarn [Kan10].

$$\text{weft:}(\delta x, \delta y) = (c_f(2\text{fract}(N_\xi(u_w - s_f\sigma^u)) - 1, c_y\sigma^v) \quad (3)$$

$$\text{warp:}(\delta x, \delta y) = (c_y\sigma^u, c_f(2\text{fract}(N_\xi(v_w - s_f\sigma^v)) - 1) \quad (4)$$

Equations 3 and 4 show the changes made to the x- and y-coordinates of the normal. The z-coordinate of the perturbed normal is generated by calculating $\sqrt{1.0 - \delta x - \delta y}$. This achieves yarn based lighting by taking into account different user-defined parameters including: fiber curvature (c_f), yarn curvature (c_y), slope of fibers (s_f), offsets in yarn space (σ^u and σ^v), and number of fiber strands (N_ξ) used to make up each yarn. The variables u_v and v_w are texture coordinates of the model, and the function fract() calculates the fraction component of a floating point.

Figure 3 shows a fabric model with weaving pattern and yarn geometry. The image on the left of Figure 3 shows a rendering of the fabric seen from a distance, with the microscale details well below an individual pixel size. The results are not significantly different to the version of rendering without yarn geometry as shown in Figure 2. The difference between the two fabric models becomes more visible when rendering a close-up view. The image on the right of Figure 3 illustrates that each yarn is constructed by several threads, as specified by the variable N_ξ, with a high yarn curvature resulting in large shaded areas on the sides of the yarns.

4 DESIGN

Fabric rendering techniques often suffer from strong aliasing effects if the resolution of the fabric microstructure is higher than the screen resolution it is displayed on. While post-processing can soften these effects, the solution is inefficient and artifacts can remain, especially for interactive applications such as cloth simulations. Some solutions use large weave sizes to reduce aliasing effects, but this is not suitable for fashion applications where realism is essential. We analyzed existing cloth rendering techniques and found that the method from Kang [Kan10] is the most promising one [YW11]. A major advantage of this algorithm is its speed, which was shown to be only 1.846 time more expensive than Gouraud shading [Kan10]. The approach also displays a high level of realism, as the results of rendered woven fabrics look realistic in close-up. However, this approach lacks a coloring scheme at the yarn level to render fabrics such as denim, and it also displays blatant aliasing artifacts on the fabric surface.

4.1 Level-of-detail Fabric Rendering

We propose a level-of-detail design for the fabric model proposed by Kang [Kan10], which removes unnecessary detail, and only renders the detailed fabric structure in close-up views.

Our design consists of two levels of visually perceivable fabric details: weave structure and yarn geometry. The visibility of yarn geometry is determined by the mipmap level, which is calculated with the use of derivative functions on texture coordinates. We explicitly render two mipmap levels for the general weave structure and the underlying yarn geometry. We limit the mipmap levels to between 0 and 1, and uses it as an alpha value for blending between the two layers of mipmap. This concept is shown in Figure 4, for those fragments that are highlighted in lighter colors, they are rendered with the general weave structure, whereas for those that are highlighted in darker colors, they are rendered with more detailed yarn geometry. This means that if the texture coordinates between neighboring fragments are changing quickly, then a higher level

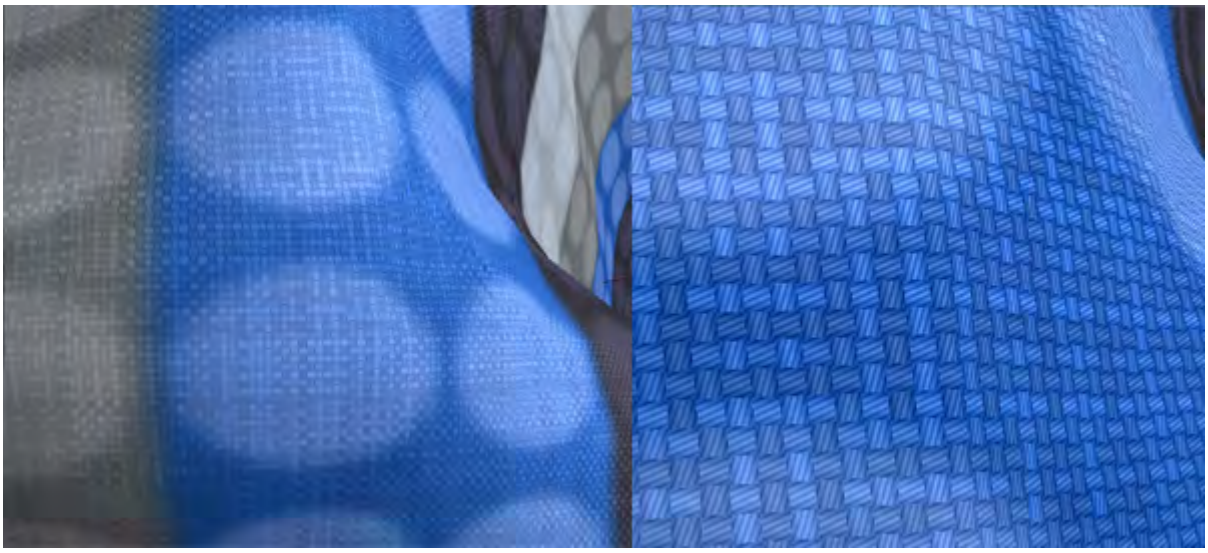


Figure 3: Rendering of weave pattern with yarn geometry seen from distance (left), and from close-up (right).

mipmap (less detail) is used, thus avoiding the rendering of unnecessary detail when the fabric is observed in a larger distance.

Figure 4: Visualization of mipmap level selection from far view point to close view point (left to right).

In essence, two MDFs are calculated, one using the pre-perturbed normals for weaving pattern rendering, and the other using the perturbed normals for yarn geometry rendering. The yarn geometry is obtained from the normal perturbation using Equations 3 and 4, which is then used as an input to the MDF. In practice, however, only the dot product between the halfway vector h and the normal vector n has to be recalculated two times for each values, with the rest of the calculations in Equation 2 only calculated once. Equation 5 shows the calculation of the final MDF, which is done by using the two previously mentioned MDFs, and weighting them with the α value obtained from the mipmap calculation that determines which level of mipmap should be used for rendering.

$$p(h) = (1.0 - \alpha)p(h_1) + \alpha p(h_2) \tag{5}$$

4.2 Extensions for Real-time applications

We also extend the model developed by Kang [Kan10] to support more types of fabrics.

4.2.1 Extended Fabric Coloring Scheme

For our system, we require the visualization of fabrics such as denim. Denim is often constructed from fiber using the twill-weaved weaving pattern. In contrast to ordinary cotton twill, denim uses different colors for the weft and warp yarn, with the most popular combination being white for the warp yarn and blue for the weft yarn.

We define extra parameters to specify the base color of individual weft and warp yarns, both in terms of diffuse and specular colors. Using these base colors, we apply Kang's algorithm [Kan10] to generate the procedural textures with weaving patterns and yarns to simulate the virtual fabrics.

4.2.2 Ambient Occlusion

The original method by Kang [Kan10] introduced an ambient occlusion term defined by the z value of the perturbed normal. Since the perturbed normal is generated to define the yarn geometry at the micro-level, the ambient occlusion term only works for scaling the anisotropic reflection at the yarn level to create a shadow effect on the reflection.

The self-shadowing effects at a higher level are not captured due to the lack of indirect lighting involved in calculating the overall reflectance of the fabric. However, self-shadowing is common in practice, e.g. when cloth is folded. Hence, we use Screen-Space Ambient Occlusion (SSAO) [Mit07] as a real-time ambient occlusion method to introduce self-shadowing effects to the existing fabric rendering method.

In the initial pass, we render our fabric at the same time as normal buffer and position buffer to avoid rendering the same object in multiple passes. We store fabric rendering results in a color buffer, and the color buffer is referred to for scaling its values using the calculated

ambient occlusion from the normal buffer and position buffer.

$$ao = max(0.0, \frac{dot(N,V)}{1.0+d}) \quad (6)$$

Equation 6 describes the calculation of the ambient occlusion of a pixel. A point (occluder) occludes another point (occludee) if the dot product between the normal of the ocludee and the vector from the ocludee to occluder is greater than zero, i.e if the point is located at the front face of the occludee, then it contributes some amount of occlusion scaled by the dot product and the distance between two points. In order to calculate the ambient occlusion at each pixel of the screen, neighboring pixels are sampled randomly using a noise function and the ambient occlusion value is averaged according to the number of sample points [Mit07].

4.2.3 Anti-Aliasing

The original implementation of the renderer produced clearly visible aliasing effects and moire patterns due to high frequency textures. These artifacts are caused by the microscopic details of the procedural textures generated in real-time. When the object is in motion, the aliasing artifacts become even more visible to temporal aliasing.

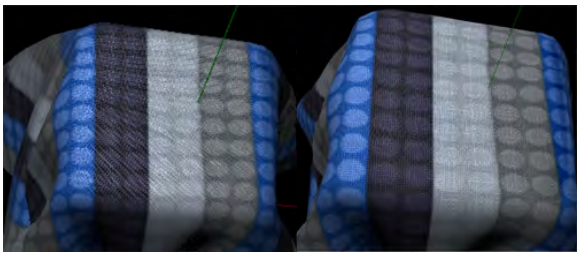


Figure 5: Aliasing of weave patterns. Comparison of fabric being viewed from a distance (left) and from close-up (right). The fabric was rendered with the original implementation by Kang [Kan10] without any anti-aliasing or level-of-detail algorithm

The left image of Figure 5 displays the distortion of weaving patterns when the viewpoint is located far away from the object. The moire patterns on the surface of the fabric are clearly visible as distorted curve lines. The fabric on the right in Figure 5 shows the weaving pattern when the viewpoint is at a closer distance to the object - no moire pattern is visible. Another example is given by Figure 6, which shows a denim fabric rendered without any underlying textures, but using blue colored weft yarns and white colored warp yarns. When the fabric is viewed from a distance (left image of Figure 6), the aliasing artifacts are more visible with this fabric due to the highly contrasted yarn colors between weft and warp yarns, also causing moire patterns to appear on the surface of the fabric. Furthermore, the twill-weave on the denim fabric is clearly distorted and unrecognizable from this distance. The aliasing artifacts are significantly reduced on the right of Figure 6, but still exist in high frequency areas such as regions where the fabric is bent around the underlying cuboid object.

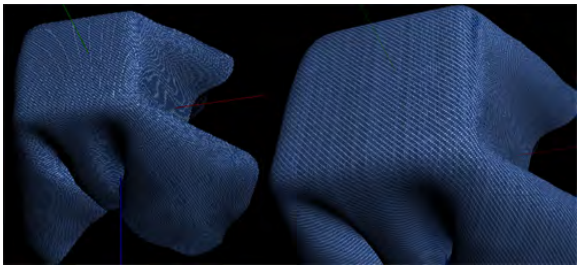


Figure 6: Aliasing of weave patterns of denim fabric. Comparison of distant viewpoint in magnified view (left) and close viewpoint (right)

An anti-aliasing method is required to reduce the moire effects on the surface of the fabric. While a level-of-detail scheme was proposed in Section 4.1, the size of the weaving patterns still introduces a high level of texture frequency on the fabric surface.

Traditionally, oversampling methods such as supersampling anti-aliasing (SSAA) and its variation, multisampling anti-aliasing (MSAA) were used to handle aliasing for graphics applications and computer games. SSAA is known to incur large bandwidth and shading cost, due to multiple numbers of samples being taken inside each pixel [HA90, Mam89], while MSAA improves on SSAA's performance by only evaluating each fragment value once and only supersampling depth and stencil values [Ake93].

Recently, post-processing anti-aliasing methods have become more popular for reducing aliasing artifacts. Methods such as morphological anti-aliasing (MLAA) [Res09] and fast approximation anti-aliasing (FXAA) [Lot09] have the advantage that they are independent to the rendering pipeline. The methods are applied at the last stage as an image-based post-processing technique. While MLAA resolves many edge aliasing problems, it is unable to handle pixel-sized features, and fails to reduce the moire-effects from high frequency textures [Res09]. FXAA employs a similar anti-aliasing procedure, where the image is used to detect edges using highly contrasting areas of each pixel, with an additional sub-pixel offset from the detected edge for low-pass filtering to achieve anti-aliasing in the sub-pixel level [Lot09]. Therefore, FXAA can be considered as a sub-pixel anti-aliasing technique and is hence potentially useful for our fabric shader. However, MSAA is the preferred choice due to its proven anti-aliasing performance over the entire object surface.

Despite the popularity of these methods, we found that they did not alleviate the high frequency aliasing prob-

lem we faced with texture aliasing from our implementations. Therefore, we decided to simply use an adaptive prefiltering approach [Ros05] inside our GLSL shader for averaging the pixel with its neighbors. The filter is adaptive such that the number of surrounding colors it calculates depends on the degree of similarity in each iteration. This algorithm is shown in Algorithm 1, which shows an overview of the algorithm for each fragment in calculating its final color. Essentially, the final color is iterated until its difference with other colors between neighboring fragments is less than a threshold, defined by the inverse distance of the fragment from the view point.

Algorithm 1 Prefiltering for fabric shader

```
count ← 1
ddx ← dFdx(worldPos) * 0.5
ddy ← dFdy(worldPos) * 0.5
while true do
    lastColor ← color
    color ← color + calcFragmentColor(worldPos
+ ddx * rand() + ddy * rand())
    count ← count + 1
    if count > 5 then
        δcolor ← lastColor − color
        if length(δcolor) < (1 / viewDistance) then
            break
        end if
    end if
end while
finalColor ← color/count
```

5 RESULTS

This section evaluates the effectiveness and efficiency of the improvements proposed by us. The following tests were performed:

- LOD fabric rendering quality test
- LOD fabric rendering performance test
- Denim Rendering

With rendering quality, we compare and contrast the quality of several images with real fabrics. To compare the effects of using level-of-detail rendering, we compare rendering results with and without the improvements, and we also compare their effects on aliasing artifacts. For rendering performance we compare the frame rates achieved with the different implementations. All tests were performed on a PC with Intel Core i7 2600k,12 GB memory at 1600 MHz with an AMD Radeon HD 6950 graphics card with 2GByte GPU memory.

5.1 Level-of-Detail Rendering

5.1.1 Rendering Quality

The level-of-detail rendering of woven fabric was tested by comparing the rendering quality of fabrics with and without our level-of-detail fabric rendering algorithm. Figure 7 shows that without level-of-detail rendering (left) many aliasing artifacts are seen, which they are not visible using level-of-detail rendering (right). The observations are confirmed by a second example shown in Figure 8, which represents a red twill-weaved woven fabric.

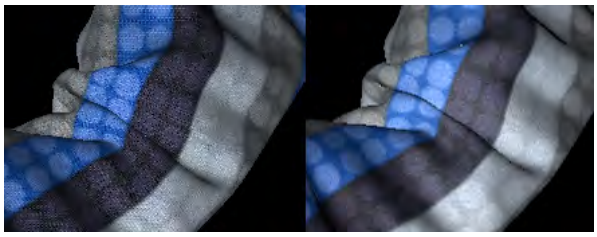


Figure 7: Level-of-detail rendering, without LOD (left) and with LOD (right).

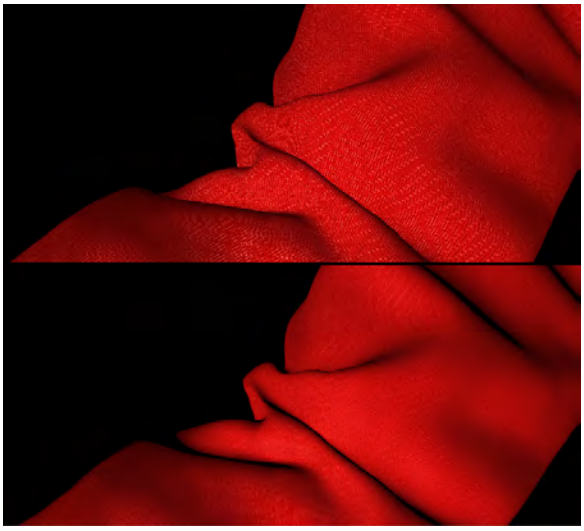


Figure 8: Level-of-detail rendering, without LOD (top) and with LOD (bottom).

The denim fabric still displays some aliasing artifacts with high frequency weaving pattern that is rendered on polygons facing away from the screen. This is due to the high contrast in color in the underlying weaving construct, coupled with the projection of weaving pattern to a much smaller projected area, thus making it difficult to smooth the high frequency changes in color at the micro-level. An example of this problem is shown in Figure 9, where the left image is a magnification of the small rectangle section in the right image. The left image illustrates some aliasing artifacts close to the edge of the fabric, with white lines going against the flow of the twill-weaved weaving pattern. These artifacts are not clearly noticeable in static images, but they

Implementation	Performance (ms)	Std. dev (ms)
Original	5.9302	0.0346
LOD	4.4037	0.04115

Table 1: Table comparing performance and standard deviation of the two algorithms in milliseconds per frame.

become very conspicuous in temporal aliasing when the fabric is in motion.

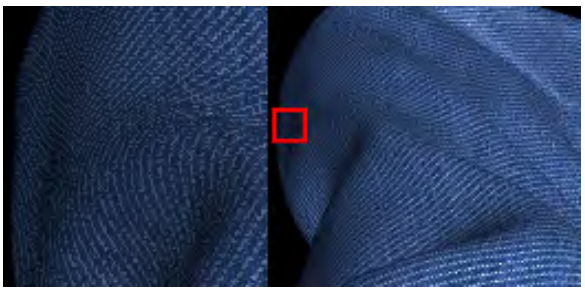


Figure 9: An example of aliasing problem due to texture projection. The left image is a magnification of the red square in the right image.

5.1.2 Performance Analysis

The performance of the two algorithms (the original algorithm and the LOD algorithm) was analyzed using an identical camera position and view, cloth model, texture, and input parameters for the rendering model. Both algorithms were tested along with the prefiltering approach, as we have decided to incorporate it to reduce the procedural texture aliasing.

Table 1 shows the results of the two algorithms, performed with a 3D model as shown in Figures 7 and 8, which contains 18892 vertices and 37566 triangles. Our level-of-detail fabric rendering algorithm is faster than the original algorithm. Given the improved rendering quality and reduced artifacts our new approach is hence preferable.

5.2 Denim Rendering

By extending the model to allow the specification of coloring of yarns, we managed to procedurally generate fabrics such as denim using the twill-weaved weaving pattern coupled with blue colored weft yarns and white colored warp yarns. Figure 10 provides a comparison between our results and a close-up of real worn denim fabric. The rendering results look realistic in terms of fabric structure, but more randomness and tear and wear needs to be incorporated into our model in the future.

We adopted the noise function proposed by Kang [Kan10]. The noise function generates random illumination at different yarns, which enhances the realism of our rendering. Note that some white warp yarns look brighter than others as is the case for real denim. So far our framework does not simulate washed denim fabric

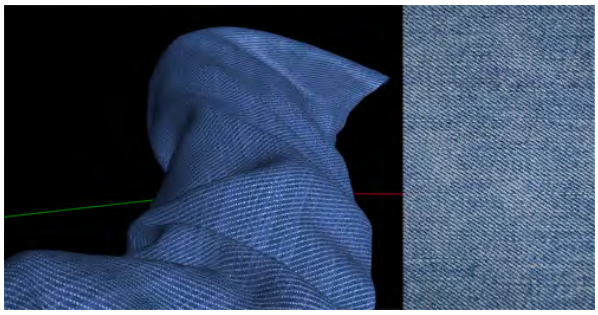


Figure 10: Denim fabric rendered with our framework (left) and a photo of real denim (right).

and hence we cannot replicate the random whitish patches in the image of real denim fabric.

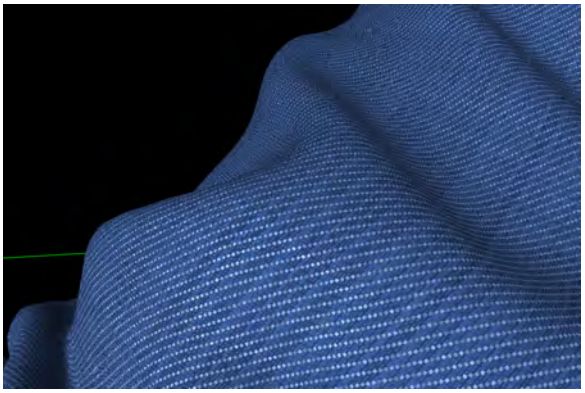


Figure 11: Close-up view of denim fabric rendered with our framework.

Figure 11 shows a close-up view of our rendering results. Our model renders the twill-weave too uniformly, lacking the natural imperfection found in real denim. When the denim is being viewed from a larger distance, the yarn geometry and weaving pattern gets aggregated and only the dominating color in the weaving pattern is visible to the user.

Figure 12 shows another close-up appearance of the denim fabric using a model of a pair of jeans.. The weaving pattern is defined to closely simulate the structure of a pair of real jeans. In order to render jeans realistically, we modified our fabric shader so that it supports the input of wash maps. Wash maps are used to define the patterns of washes for a pair of jeans, where washes are patterns of white patches on the jeans as shown in Figure 12. In Figure 12, a pair of rendered jeans is shown on the left with a pair of real jeans on the right. This close-up view demonstrates that the weaving pattern of the rendered jeans closely resembles the weaving pattern of the real jeans, as they are both similar in structure and size relative to the jeans model. Furthermore, at this viewing distance, the appearance of both jeans is very similar to each other.

A comparison of an entire pair of real jeans and our rendered jeans using a distance view is shown in Figure 13.

Figure 12: Jeans comparison when viewed from close-up: rendered jeans (left) and real jeans (right).

The rendered jeans (left) in Figure 13 closely resembles the real jeans (right) in Figure 13. From this distance, the weaving pattern is completely invisible to the observer, and aliasing artifacts are also unrecognizable on the fabric surface with the use of our LOD algorithm and prefiltering approach.

Figure 13: Jeans comparison when viewed from a distance: rendered jeans (left) and real jeans (right).

In our results, we found that that the use of direct lighting often makes the resulting rendered object too bright in areas that are occluded by the object itself. An example is shown in Figure 14, where the left image is the rendered jeans without ambient occlusion, and the right image is the rendered jeans with ambient occlusion. In this scene, the light is positioned to the left of the jeans, hence the inner thigh area should be dark as it is not directly illuminated by the light source. However, without ambient occlusion the rendered jeans still seems to be too bright around this area, and we found that the SSAO approach results in a more natural appearance of occluded areas.

Figure 14: Effect of ambient occlusion, before ambient occlusion (left), and after ambient occlusion (right)

6 CONCLUSION

In this paper, we analyzed several existing fabric rendering techniques. We chose to use the method proposed by Kang [Kan10] as a basis for our fabric shader, due to its rendering quality and performance. Several extensions were proposed to improve the robustness of the model and for supporting fabrics such as denim, and

ambient occlusion for enhancing the realism of self-occlusion of the cloth model. Furthermore, we proposed a level-of-detail approach in visualizing the aggregation of details with the use of a mipmap LOD selection mechanism, to help reduce aliasing artifacts resulting from high frequency textures. Overall, our extension to the model enabled us to successfully render denim fabric with an unwashed look and it significantly reduced aliasing problems. With the incorporation of wash maps to specify areas of washes, we have successfully replicated the overall and close-up appearance of actual jeans.

7 FUTURE WORK

The weave-based level-of-detail algorithm only reduces parts of the aliasing caused by high frequency textures. It still suffers from aliasing from small scaled weaving pattern and highly contrasting weft and warp yarns' colors, such as for denim fabric, depending on the size of weft and warp segments. Our approach rectified the aliasing problem that is often seen in weave-based fabric rendering approaches, but a better algorithm can be investigated in the future to reconstruct the appearance of high-detail level from lower levels, rather than filtering these details away.

8 REFERENCES

[Ake93] Kurt Akeley. Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 109–116, New York, NY, USA, 1993. ACM.

[AMTF03] N. Adabala, N. Magnenat-Thalmann, and G. Fei. Real-time rendering of woven clothes. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 41–47. ACM, 2003.

[AS00] M. Ashikhmin and P. Shirley. An anisotropic phong BRDF model. *Journal of graphics tools*, 5(2):25–32, 2000.

[DLH01] K. Daubert, H.P.A. Lensch, and W. Heidrich. Efficient cloth modeling and rendering. In *Rendering techniques 2001: proceedings of the Eurographics workshop in London, United Kingdom, June 25-27, 2001*, page 63. Springer Verlag Wien, 2001.

[HA90] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. *SIGGRAPH Comput. Graph.*, 24(4):309–318, September 1990.

[Ira08] Piti Irawan. *Appearance of woven cloth*. PhD thesis, Ithaca, NY, USA, 2008. AAI3295837.

[Kan10] Y.M. Kang. Realtime rendering of realistic fabric with alternation of deformed anisotropy. *Motion in Games*, pages 301–312, 2010.

[Kau05] J. Kautz. Approximate bidirectional texture functions. *GPU Gems*, 2:177–187, 2005.

[Lot09] T. Lottes. FXAA. 2009. Also available as `http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf`.

[Mam89] Abraham Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl.*, 9(4):43–55, July 1989.

[Mit07] Martin Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[Res09] A. Reshetov. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 109–116. ACM, 2009.

[Ros05] R.J. Rost. *OpenGL (R) shading language*. Addison-Wesley Professional, 2005.

[War92] G.J. Ward. Measuring and modeling anisotropic reflection. *ACM SIGGRAPH Computer Graphics*, 26(2):265–272, 1992.

[WZT+08] J. Wang, S. Zhao, X. Tong, J. Snyder, and B. Guo. Modeling anisotropic surface reflectance with example-based microfacet synthesis. In *ACM SIGGRAPH 2008 papers*, pages 1–9. ACM, 2008.

[YW11] W. Yuen and B. Wünsche. An evaluation on woven cloth rendering techniques. In *Proceedings of the 26th International Image and Vision Computing New Zealand Conference (IVCNZ 2011)*, pages 7–12, Auckland, New Zealand, November 2011. Also available as `http://www.cs.auckland.ac.nz/~burkhard/Publications/IVCNZ2011_YuenWuensche.pdf`.

[YYTI92] T. Yasuda, S. Yokoi, J. Toriwaki, and K. Inagaki. A shading model for cloth objects. *Computer Graphics and Applications, IEEE*, 12(6):15–24, 1992.

[ZJMB11] S. Zhao, W. Jakob, S. Marschner, and K. Bala. Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph*, 30(44):1–44, 2011.

Error Metrics for Smart Image Refinement

Julian Amann
amannj@in.tum.de

Matthäus G. Chajdas
CG/Vis Group, CS Departement
Technische Universität München
Boltzmannstrasse 3
85748 Garching, Germany
chajdas@tum.de

Rüdiger Westermann
westermann@tum.de

ABSTRACT

Scanline rasterization is still the dominating approach in real-time rendering. For performance reasons, real-time ray tracing is only used in special applications. However, ray tracing computes better shadows, reflections, refractions, depth-of-field and various other visual effects, which are hard to achieve with a scanline rasterizer. A hybrid rendering approach benefits from the high performance of a rasterizer and the quality of a ray tracer. In this work, a GPU-based hybrid rasterization and ray tracing system that supports reflections, depth-of-field and shadows is introduced. The system estimates the quality improvement that a ray tracer could achieve in comparison to a rasterization based approach. Afterwards, regions of the rasterized image with a high estimated quality improvement index are refined by ray tracing.

Keywords

hybrid rendering, reflection error metric, depth-of-field error metric

1 INTRODUCTION

Nowadays, rasterization based graphic pipelines dominate real-time 3D computer graphics, because graphics hardware is highly optimized for this rendering algorithm. Many years of research have been spent on developing massive parallel processing units that are able to process complete pixel quads in a fast way by exploiting frame buffer locality and coherence [KMS10].

Even though rasterization has a lot of advantages, it also has some limitations. It performs well evaluating local illumination models, however there are problems with global effects like reflections. Because rasterization is limited to local illumination models, it is hard to compute physically correct reflections of the environment. For that reason, approximations like environment maps [Gre86] are used, which can result in visually plausible reflections.

Shadows are also a very challenging problem for rasterization. Although there are numerous shadow mapping and shadow volume techniques, they all have some inherent problems that arise from the local view of shading in the rasterization process. For example, most shadow mapping techniques suffer from the well-known biasing problems or have shadow mapping artifacts due to a too small shadow map resolution [ESAW11]. The number of annually appearing publications to shadow topics proves that the generation of shadows is still a challenging subject.

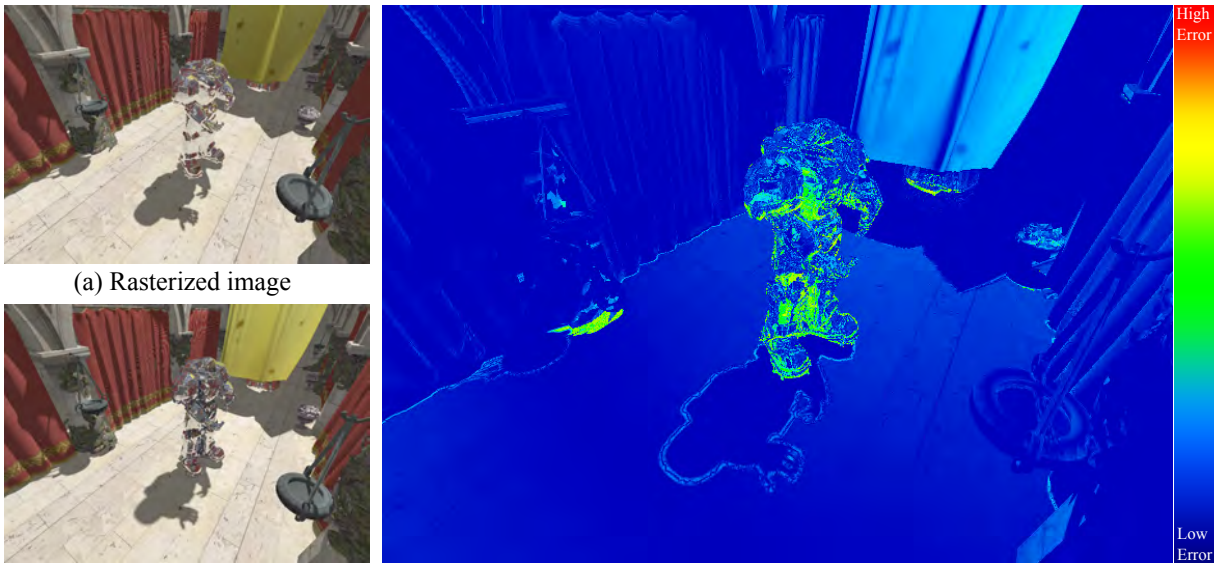
Besides reflections and shadows, it is also hard to simulate a correct thin-lens camera with rasterization. Most depth-of-field techniques which are rasterization based compute the circle of confusion and then just blur the image with the computed radius, which results in an incorrect depth-of-field effect [Pot81].

Secondary effects, like shadows or reflections are hard to achieve with a rasterization approach. A ray tracer can natively handle shadows and multiple reflections just by sending additional secondary rays. With a ray tracer it is not hard to simulate these effects. A thin-lens camera model can also be easily implemented in a ray tracer to get a nice depth-of-field effect.

2 MOTIVATION

Because ray tracing is computationally very expensive, rasterization is still used for games today. Another reason is that under some circumstances a rasterizer can produce exactly the same image as a ray tracer could. Figure 1 compares a scene rendered with ray tracing to a scene rendered with rasterization.

Remarkable is the fact that the rasterized image took about 7 ms to render with precomputed environment maps on commodity hardware (NVIDIA GeForce GTX



(a) Rasterized image

(b) Ray traced image

(c) Difference image

Figure 1: A difference image of scene (a) that has been rendered with rasterization (b) and ray tracing (c). The temperature scale on the left shows how to interpret the colors of the difference image. Blue colors mean a low difference and red colors mean a high difference. For example high differences can be seen in the reflecting object in the center of scene.

460) at a resolution of 762 x 538 pixels with no scene graph optimizations. However, the ray traced image with the same image quality and resolution and a highly optimized scene graph took roughly 976 ms to render (the performance measurements were made with the render system described in section 7).

Being given the choice of those techniques, one has to weigh the speed up against the high image quality. In order to profit from the advantages of rasterization and ray tracing, a hybrid approach is preferable. For example, a hybrid technique can just use ray tracing to compute reflections while the rest of the scene can be conservatively rasterized. This helps to get high quality reflections at the cost of only ray tracing the parts of the scene that have reflecting materials.

Ideally, one would estimate beforehand how big the difference between the rasterized and ray traced image is. This difference (see Figure 1c) could be used as a hint to find out where it is most appropriate to send some rays to improve the image quality.

3 CONTRIBUTIONS

This paper describes an error metric for reflections, shadow mapping and depth-of-field, which estimates the expected pixel error between the rasterized and the ray traced scene. The error metric is a heuristic one, which yields an expected but not an absolutely correct error value. During the rasterization of the scene, the error value is calculated and stored in an additional render target. After a first approximation of the current scene by rasterization, the error value of each pixel can be used to find out where it is most appropriate to refine the rasterized image by ray tracing.

This paper also presents a scheduling strategy that determines in which order the parts of the image are getting refined via ray tracing, so the scene converges quickly. This means that parts of the scene with high error get ray traced first and more often than parts of the scene with a low error. This helps to prevent wasting a lot of computation time on parts of the image where no difference or only a slight difference between the ray traced and the rasterized version is observable.

Furthermore, the implementation of a hybrid rasterization and ray tracing framework that is based on Microsoft Direct3D 11, NVIDIA CUDA Toolkit and NVIDIA OptiX 2.5[1] ray tracing engine is described, which demonstrates the feasibility of the described smart image refinement system. The system is called smart, because it refines the image according to the error estimate.

4 RELATED WORK

In the following section related work is briefly presented. Contributions made by other researchers reach from simple hybrid rendering systems to sophisticated perceptually-based techniques.

[1] NVIDIA OptiX is a freely available low level ray tracing engine that runs entirely on the GPU. Currently OpitX is only supported by NVIDIA GPUs. Similar as Direct3D or OpenGL provides an interface to an abstract rasterizer which can be used to implement various rasterization-based algorithms, OptiX provides an interface to an abstract ray tracer.

Perceptually-based techniques try to shortcut the render process by computing a perceptually indistinguishable solution instead of a fully converged one. In [YPG01], a perceptually-based technique is described that calculates a spatio–temporal error tolerance map. The computation of the error map takes a few seconds and is targeted at offline renderers. Each pixel of the error map indicates how much effort should be spent on the respective pixel. The error value is determined by harnessing knowledge about the human visual system and a model which predicts visual attention. For example, the eye is less sensitive to areas with high spatial frequency patterns or movements. Alongside with a prediction on where the viewer directs his or her attention, an estimate is computed that describes how important a pixel will be. This estimate is saved in the error map and is used during the render process to spend more time on important regions of the image. The paper's authors achieved on their test rendering system a $6\times$ to $8\times$ speedup.

[Cab10] uses a simple error metric. The metric consists of only a binary decision if ray tracing or rasterization is to be used. If the rasterizer renders a triangle with a transparent or reflecting material, a flag is set in a ray casting buffer. Afterwards all pixels marked with the flag get ray traced and combined with the rasterized image. They use a CPU-based ray tracer.

In [KBM10], a hybrid approach is shown that combines shadow mapping and ray tracing to render shadows. In a direct render pass and from a small number of shadow maps that are used to approximate an area light source by several point lights, a shadow refinement mask is derived. The mask is used to identify the penumbra region of an area light source. A pixel is classified as inside the penumbra when it cannot be seen from all point lights. Afterwards, the penumbra pixels are dilated by a 5×5 structuring element. The dilated region is then rendered by a CPU-based ray tracer to compute accurate shadows.

5 ERROR METRICS

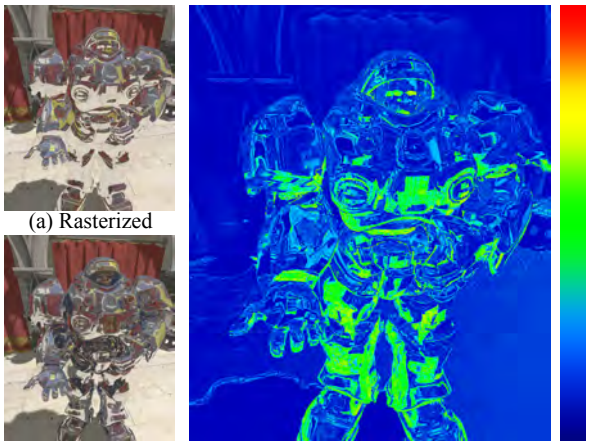
This section describes different error metrics for reflections, depth-of-field and soft shadows. The presented error metrics are used by the smart image refinement system to find out in which regions refinement by ray tracing is most appropriate. An error value is estimated for each pixel and is stored in an additional render target. The error metric is used as a heuristic that indicates how likely the calculated pixel is wrong in comparison to a pure ray traced version of the scene.

A high error value indicates that the approximation by the rasterizer contains a high error, whereas a small error value indicates low errors in the approximation by the rasterizer. The error value is just based on heuristics, which means that in certain circumstances, a high error value refers to a pixel that has only a small real or no approximation error at all compared to the ray traced scene. Conservative error metrics were chosen, so no pixels get estimated as correctly approximated, even if they are not correct.

Each error metric is normalized, which means it generates an error value in the range of $[0,1]$.

Reflections

Reflections can be approximated by environment maps in a rasterization based environment. Figure 2 compares reflections rendered by rasterization to reflections rendered by a recursive ray tracer.



(a) Rasterized (b) Ray traced (c) Difference

Figure 2: The rasterized image (a) approximates reflection with an environment of the scene. Figure (b) shows the same scene rendered with a recursive ray tracer. The difference image (c) visualizes the difference between Figure a and b. A red value indicates a high difference and a blue value a small difference.

As can be seen from the difference image, the approximated rasterized image is far from perfect. It contains several regions with wrong reflections.

The simplest to think of error heuristic is one that just makes a binary decision, depending on the criterion if a fragment is part of a reflection object or not [Cab10]. Assuming it is part of the reflecting material, the error metric returns 1, in all other cases it returns 0:

$$E_{reflection_1} = \begin{cases} 1 & : \text{reflecting material} \\ 0 & : \text{else} \end{cases}$$

The previous classification is a very simple one. A more sophisticated error metric can be derived, if we try to understand why the approximation of an environment is wrong. Figure 3 shows two major problems of environment mapping.

Put the case that we want to rasterize a reflecting sphere with the help of environment mapping. For a point P

on the sphere (see Figure 3) we need to look up the current reflection from the environment map. For the look up in the environment texture, we first need to compute the reflection vector. The look up vector is then transformed to texture coordinates which are afterwards used to access the reflection map. The problem with this approach is that the environment map has usually been rendered from the center of the reflecting object. This means, we get the color that is seen from the center of the environment map towards the direction of the reflected vector. Instead of this color, the correct color would be the color that can be seen from the point P towards the direction of the reflected vector. Figure 3 illustrates this. From point P, the reflection direction points toward the colored sphere (r_2). So we expect to see a colored sphere in the reflection of P. But in fact, the environment map technique uses the center of the environment map to look up the color of the reflected object. Looking from the center of the environment map into the direction of the reflection vector, a yellow cube can be seen instead of a colored sphere.

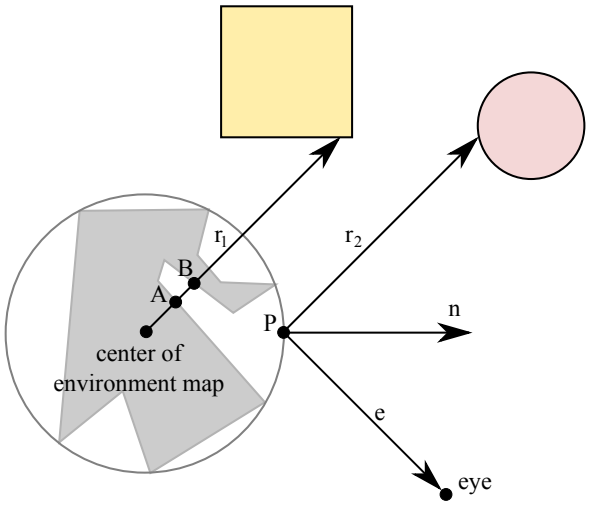


Figure 3: Environment mapping has two major problems: First of all each reflection is computed as if the reflecting point (P) would lie in the center of the environment map. Also it cannot handle self-reflections, which leads to incorrect shading results at point A.

The environment map technique would result in a correct shading, if the shaded point is located directly in the environment map center. For points that only have a very small distance to the environment map center, this approximation works as well. But for points with a further distance to the environment map center, the approximation by an environment map gets worse.

From this observation, a simple heuristic can be derived: The further a point is away from the environment map center, the more likely an environment map technique results in an incorrect approximation. This means that the distance between a point (p) of a reflecting object and the environment map center (c) needs to incorporate in the approximating environment map error metric:

$$E_{reflection_2} = \begin{cases} \frac{distance(p,c)}{maxDistance} & \text{: reflecting material} \\ 0 & \text{: else} \end{cases}$$

Another error metric can be deduced from the incident vector and reflection vector, as Figure 4 illustrates. Assuming that there is a scene with a reflecting sphere where the center of the environment map has been placed at the center of the sphere, this would mean that the environment map has been rendered from the center of the sphere. Looking at the reflecting sphere in a way that the incident ray (the ray from the eye point to a point in the scene) is hitting directly the center of the sphere, as this is the case for the incident vector I_1, the look up in the environment map will yield the correct reflection color.

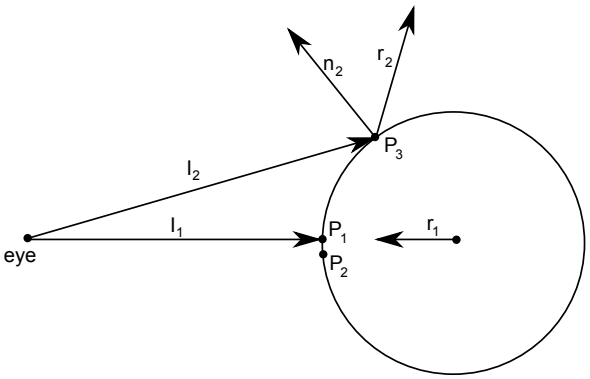


Figure 4: Incident and corresponding reflected vectors

The returned reflection color is correct because the given incident vector I_1 leads to the reflection vector r_1, which means we want to know what can be seen from the intersection point P_1 into the direction r_1. But this is exactly the same question as what can be seen from the center of the environment map into the direction of r_1. If we look from the eye point into the direction of a point that is near to P_1 like point P_2, the incident vector narrowly misses a hit with the center of the environment map, but the looked up direction in the corresponding environment map is approximately not too far from being correct.

It seems that for small angles between the incident and the reflection vector, the approximated reflection vector is almost correct, but for bigger angles like the angle between incident vector I_2 and r_2 it gets worse. From this property, the following error heuristic can be derived:

$$E_{reflection_3} = \begin{cases} \langle -i, r \rangle & \text{: reflecting material} \\ 0 & \text{: else} \end{cases}$$

It is assumed the reflection vector r and the incident vector (vector from eye point to shaded point) i in the

above equation are normalized. The angle between the vector r and $-i$ is always in the range $[0°, 90°]$. Since the angle can be in the range $[0°, 180°)$ the dot product fails for greater than 90° angles. To circumvent this problem instead of considering the reflected vector the normal can be considered which leads to the following equation:

$$E_{reflection_4} = \begin{cases} \langle -i, n \rangle & : \text{reflecting material} \\ 0 & : \text{else} \end{cases}$$

This works because the angle between the incident and reflected vector is directly proportional to angle between the reflected and the incident vector. The angle between the negative incident vector and the normal can never exceed 90°.

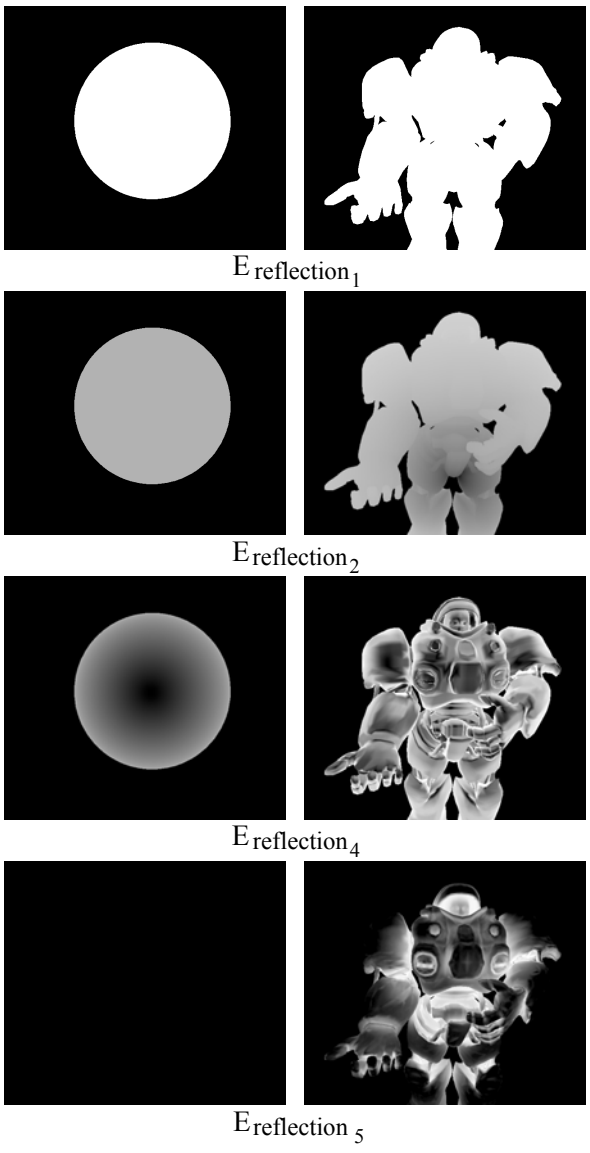


Figure 5: Displays the different reflection error metrics applied to a scene with sphere (left) and a scene with a more complex shape (right)

Another not yet considered problem of the error metric that is also related with environment maps are self-reflections. Concave objects are displayed incorrectly by an environment map technique. Figure 3 shows the reason for this. Assuming we want to compute the reflection of the point A in Figure 3 given the reflection vector r_1. In a look up in the environment, the yellow color from the yellow cube is returned. However in fact the reflection ray intersects the reflecting object itself (a so-called self-reflection) in point B and despite of this, the yellow color from the environment map is nonsense. Self-reflections can probably not be handled by environment maps. We can take care of this in our error metric by using an ambient occlusion map. The ambient occlusion map can be interpreted as a description of the curvature of the reflecting object. This information can be directly used in a heuristic that estimates the possibility of self-reflections:

$$E_{reflection_5} = \begin{cases} k_a(p) & : \text{reflecting material} \\ 0 & : \text{else} \end{cases}$$

$k_a(p)$ refers here to the ambient occlusion term.

Figure 5 shows the different error metrics applied to two sample scenes.

Depth-of-Field

Most rasterization based depth-of-field techniques are image based and use only the depth buffer and color buffer to compute a depth-of-field effect. Thereby, the information about the scene is lost. In a ray tracer, the lens is sampled at different locations. Each sample on the lens has a different view of the current scene. In the rasterizer approach, we have only one view at the scene from the center of the lens. This can lead to missing objects, because in some cases from some points on the lens, objects can be seen that cannot be seen from the center of the lens.

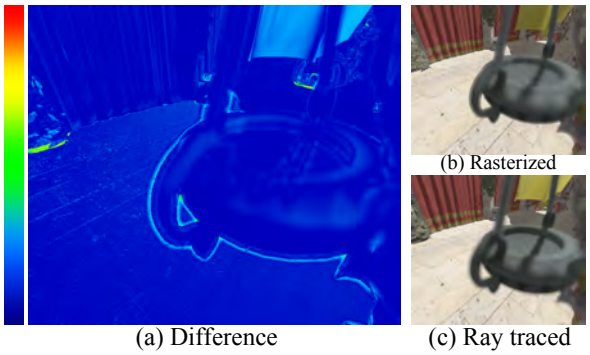
Another problem of most depth-of-field techniques is color leaking. Color leaking can be seen around sharp edges that are in focus in which other blurry objects from the background bleed into [LU08]. The other way around, objects in the foreground can bleed into objects in the background. Figure 6 shows this effect.

As demonstrated in Figure 7, rasterization based depth-of-field have problems in regions with high depth discontinuities. This knowledge can be exploited to construct an error metric for the depth-of-field.

To find depth discontinuities, an edge filter, like the Sobel filter, can be applied. A problem with this approach is that the founded edges have to be dilated by a structuring element, since the artifacts do not only occur at the identified edge pixels, but also in the neighborhood of the edge pixel according the circle of confusion. The

(a) Rasterized (b) Raytraced

Figure 6: A blurry foreground object bleeds into the focused background object.



(a) Difference (b) Rasterized (c) Ray traced

Figure 7: Difference image of scene (with applied depth of field effect) (a) that has been rendered with rasterization (b) and ray tracing (c). Regions with high depth discontinuities are problematic for rasterization based rendering techniques.

maximal radius of the circle of confusion C_{max} for the dilation can be determined for a point p by the following equation with image distance V_P, focal length F and aperture number n (z_p is the distance between the point p and the image plane):

$$C_{max}(p) = max(C(z_p), C_\infty)$$

$$C_\infty = \lim_{z \to \infty} C(z) = |F - V_p| \frac{1}{n}$$

For simplicity reasons, we use a quad shape structuring element in our implementation to approximate the circle of confusion. Figure 8 shows the error metric for depth-of-field.

The Error metric for depth of field can be expressed as:

$$E_{dof} = \begin{cases} 1 & : \text{Vicinity of a depth discontinuity} \\ 0 & : \text{else} \end{cases}$$

The described error metric is not absolutely conservative, which means that errors can also occur in regions

(a) Rasterized (b) Dilated

Figure 8: After the depth discontinues have been marked, they need to be dilated according to the circle of confusion.

that were not classified with an error value of 0. However, it can give a smart image refinement system a good hint where to start the refinement process.

Shadows

In [GBP06], an algorithm has been described that can be modified to estimate where the penumbra of a given area source light will be projected onto the scene. In Figure 9, the estimated penumbra region is shown in green color.

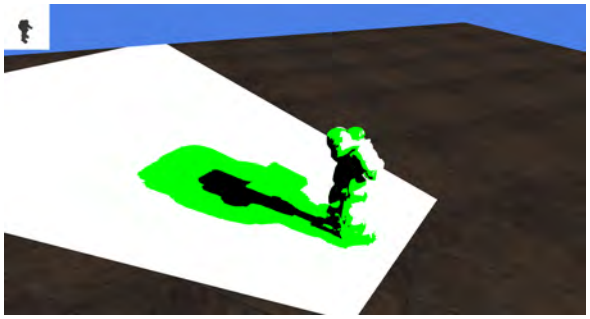


Figure 9: The estimated penumbra region is shown in green color.

The algorithm described in [GBP06] makes a conservative estimation of the penumbra region, and is therefore perfectly appropriate to be used as an error metric. The difference between the shadow generated by the rasterization system and the ray tracer is only notable in the penumbra region. In umbra regions and in regions where the area light source is not occluded by other objects (so that the scene is fully lit by the light source) no difference between the rasterizer and ray tracer is noticeable (see Figure 1 - only the penumbra region needs to be refined).

In [Mic07], an improved algorithm is presented which can estimate a tighter, conservative penumbra estimation than the algorithm described by [GBP06]. Even though it requires more memory, the tighter estimation reduces the amount of pixels that have to be refined, resulting in an overall improved performance.

The corresponding error metric for soft shadow is therefore quite simple:

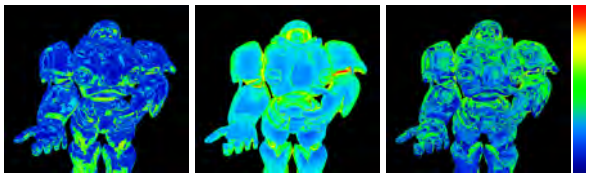
$$E_{shadow} = \begin{cases} 1 & : \text{Pixel resides in penumbra region} \\ 0 & : \text{else} \end{cases}$$

Combination of Error Metrics

The different error metrics can be combined in multiple ways. A naive idea is to calculate an averaged sum:

$$E_{combined_1} = \frac{\sum_{i=1}^{n} E_i(p)}{n}$$

Figure 10 shows the quality of the average sum metric.



(a) (b) (c)

Figure 10: Quality of averaged sum metric. (a) shows the real error, (b) the estimated error and (c) the difference image.

For a better estimation, a more complex combination is required. A simple, yet effective approach is to use a linear combination of the different error metrics (i.e. $E_{reflection_i}$, E_{dof}, E_{shadow}) and let the smart image refinement system automatically determine the coefficients λ_i by rendering multiple views of a scene with the goal of minimizing the difference between the estimated and the real error:

$$E_{combined_2} = \sum_{i=1}^{n} \lambda_i E_i(p)$$

The determination of the factors λ_i is done as a pre-process for each different scene. In this pre-process a certain number of screenshots from the scene is taken (with pure rasterization and pure ray tracing). Then a random tuple of λ_i coefficients is chosen and the combined metric $E_{combined_2}$ is then compared with the real error. We repeat this step multiple times and choose the λ_i coefficients which result in the best approximation for all test images.

6 SCHEDULING STRATEGY

This section describes how the error value is used to direct the rendering process of the smart image refinement system.

First the scene is rasterized by the rasterization system. During rasterization, an error value is also computed as described in the previous section about error metrics. After the rasterization pass, the color buffer and the error color buffer are filled. Now post-processing effects are applied to the previously rendered scenery. The post-processing result is written to the post-process color buffer; after this, the post-process error buffer is computed. Then the error buffer and the post-process error buffer get composed in a combined error buffer. For each pixel, an error value is computed and stored in the combined error buffer. After composing the error buffers, the next step is to sort the pixels. The error buffer also stores, besides the error value, the position for each pixel. The position is needed to find out to which pixel a corresponding error value belongs to after reordering them according their error values. After sorting the error pixels, they are gathered in the request buffer. Additionally to the position, a sample count value is also stored in the request buffer for each pixel that determines how many rays should be traced for the corresponding pixel. The sample count is determined by an estimation pass that fills the request buffer. After the request buffer is filled, it is handed over to the ray tracing system. The ray tracing system reads the first entry from the request buffer and samples the corresponding pixel according to the sample count. The ray tracer proceeds this process for a user-defined maximum number of pixels. After the maximum number is reached, the ray tracing process stops and the smart image refinement system continues with blending the current ray traced image with the computed rasterized image. The blending factor of each pixel depends on the total number of samples that were computed for the corresponding pixel by this time. Figure 11 gives an overview of this process. This process is repeated until the whole image is refined.

7 IMPLEMENTATION

For the implementation of the smart image refinement system Direct3D 11, CUDA 4.1 and OptiX 2.5 ([Ste10]) have been used. Direct3D is used for the rasterization part and analogously OptiX is used for the ray tracing part. Thus all rendering is done on the GPU. Optionally the system can perform pure rasterization with Direct3D or pure ray tracing with OptiX. In the case of pure ray tracing Direct3D is needed only to show the generated OptiX output buffer. Pure ray tracing and rasterization is used for comparison purposes like the time measurements in section 2 or in table 1.

The ray tracing subsystem uses a SBVH acceleration structure [SFD09] provided by OptiX to accelerate ray-triangle intersections. The rasterizer subsystem renders without any scene graph optimizations in a brute force manner.

A pixel shader is used to write the error values during rasterization to an additional render target (the error buffer). Some error values can be only determined after post-processing so there is an additional error buffer (post-process error buffer) which stores the error values determined during applying post-processing effects like depth-of-field. The combined error buffer which contains the unsorted error values is shared with CUDA.

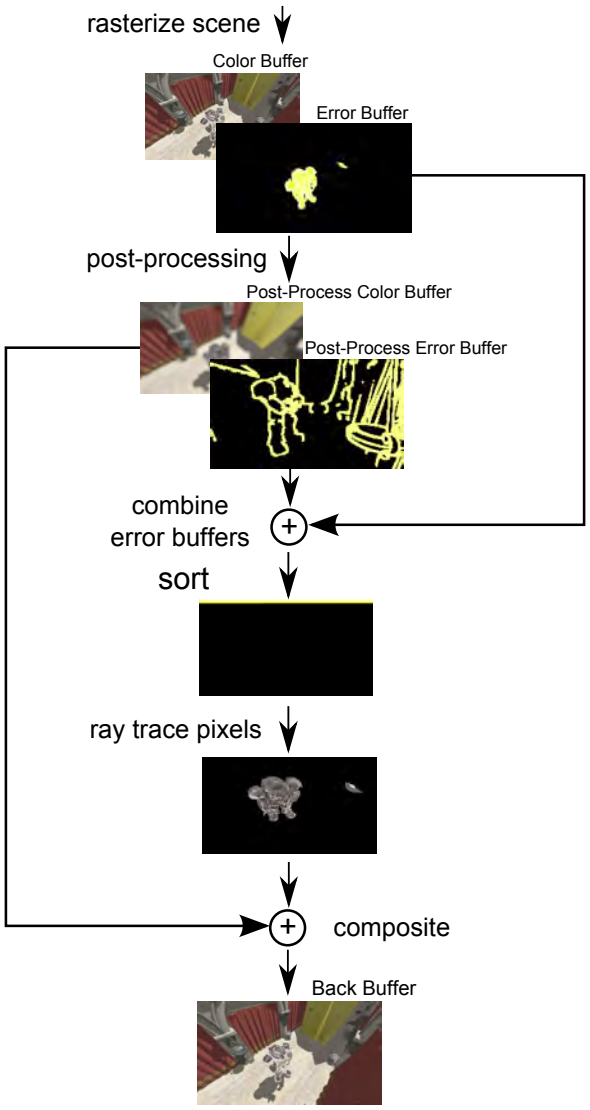


Figure 11: Overview of the smart image refinement system. During rasterization an error buffer is generated. The error buffer is sorted to give the ray tracing subsystem information where refinement makes most sense.

Thrust [HB10], a CUDA library is then used to sort all error values. The error values are encoded in such a way that the radix sort algorithm of Thrust can be used. After sorting the pixel values according their error values a full screen quad is drawn with the size of the sorted error buffer. In this step all pixel are gathered in a request buffer which is implemented as an structured buffer. The request buffer is list with the pixels that need to be refined. Since the error buffer has been sorted the list is also sorted according to the error value.

8 RESULTS

Table 1 shows a performance comparison of pure ray tracing and the prototypically implemented smart image refinement system.

Resolution	PRT in ms	SIR in ms	Error pixel (%)
800×600	440	406	188023 (39)
800×600	312	230	100934 (21)
800×600	203	79	20158 (0.4)
800×600	145	45	4181 (0.01)
1024×768	640	587	290542 (36)
1024×768	305	185	88063 (11)
1024×768	238	84	32889 (4)
1024×768	201	52	32889 (1)
1920×1080	1510	1463	805368 (39)
1920×1080	1107	901	499369 (24)
1920×1080	639	243	145239 (7)
1920×1080	484	113	44140 (2)

Table 1: Performance comparison of pure ray tracing (PRT) and smart image refinement (SIR). In the SIR implementation, one primary ray is traced for each error pixel. Also a shadow and reflection ray is cast per intersection. This is done recursively for reflections three times.

As can be seen from Table 1 smart image refinement is faster than pure ray tracing and at the same time it has the same image quality, provided that conservative error metrics are used. All measurements in this section were made with a NVIDIA GeForce GTX 560 Ti. As a test scene, the extended Atrium Sponza Palace scene that was originally created by Marko Dabrovic and extended by Frank Meinl has been chosen.

The sorting only has to be performed when the scene or camera orientation/position changes. In the implementation of the smart image refinement system a user-defined number of rays are always traced. For instance, the tracing of 8192 rays and the composition of the ray traced and rasterized image takes about 30 ms, depending on the current scene on camera view. This makes it possible to show the user first results after a short render time. Something that has be taken into consideration as well is the fact that in a pure ray traced based approach, the same number of samples is computed for each pixel, no matter if refinement makes sense for the corresponding pixel.

The performance of the smart image refinement system drops with higher error pixel rates. The major reason for this is that resources (e.g. error buffer, request buffer) have to be copied between Direct3D 11, CUDA and OptiX because they cannot be directly shared (some API extensions to improve the interoperability between OptiX, CUDA and Direct3D could avoid these copying steps). For example to hand over the error pixels that should be refined by OptiX, a request buffer has to be filled with the sorted data from a CUDA buffer. The CUDA buffer cannot be directly accessed by OptiX. The data has to be copied first.

9 CONCLUSIONS AND FUTURE WORK

In this work, a GPU-based hybrid rasterization and ray tracing system was presented that is able to direct the render processes to regions with high relevance. Regions with a high error are getting refined first and more often than regions with a small error value. This helps to converge fast to a stable image and avoids at the same time the waste of computing time in regions that do not need any refinement.

There is some scope for improvements of the described error metrics.

Besides reflections, shadows and depth-of-field, it would also be interesting to see how other effects like ambient occlusion (AO) or refractions can be integrated into a smart image refinement system. In the case of AO, a screen based ambient occlusion technique can be employed in the rasterizer to compute a fast approximation of an occlusion term.

Another interesting aspect that has not been considered in this work is global illumination. Global illumination could be approximated with light propagation volumes and refined with a more sophisticated ray tracing technique like path tracing.

There are several real-time perceptually based approaches like [CKC03] which try to cut down rendering time by focusing on important parts. These ideas can be combined with our approach.

10 REFERENCES

[Cab10] Cabeleira Joăo. Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time. http://www.voltaico.net/files/article.pdf, 2010.

[CKC03] Cater, K., Chalmers, A., and Ward, G. Detail to attention: exploiting visual tasks for selective rendering. Proceedings of the 14th Eurographics workshop on Rendering, Eurographics Association, 270-280, 2003.

[ESAW11] Eisemann, E.; Schwarz, M.; Assarsson, U. & Wimmer, M., Real-Time Shadows A. K. Peters, Ltd., 2011.

[GBP06] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by backprojection. In Eurographics Symposium on Rendering (EGSR), Nicosia, Cyprus, 26/06/2006-28/06/2006, pages 227-234. Eurographics, 2006.

[Gre86] Ned Greene. Environment mapping and other applications of world projections. IEEE Comput. Graph. Appl., 6:21-29, November 1986.

[HB10] Jared Hoberock and Nathan Bell. Thrust: A parallel template library, Version 1.3.0. http://www.meganewtons.com/, 2010.

[KBM10] Erik Knauer, Jakob Bärz, and Stefan Müller. A hybrid approach to interactive global illumination and soft shadows. Vis. Comput., 26(6-8):565-574, 2010.

[KMS10] Jan Kautz Kenny Mitchell, Christian Oberholzer and Peter-Pike Sloan. Bridging Ray and Raster Processing on GPUs. High-Performance Graphics 2010 Poster, 2010.

[LU08] Per Lönroth and Mattias Unger. Advanced Real-time Post-Processing using GPGPU techniques, Technical Report, No. 2008-06-11, Linköping University, 2008.

[Mic07] Michael Schwarz and Marc Stamminger. Bitmask soft shadows. Computer Graphics Forum, Vol. 26, No. 3, pages 515-524, 2007.

[Pot81] Potmesil, Michael and Chakravarty, Indranil. A lens and aperture camera model for synthetic image generation. In SIGGRAPH 81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques, pages 297-305, New York, NY, USA, 1981. ACM.

[SFD09] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In Proceedings of the Conference on High Performance Graphics 2009, HPG 09, pages 7-13, New York, NY, USA, 2009. ACM.

[Ste10] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison and Martin Stich. OptiX: A General Purpose Ray Tracing Engine. ACM Transactions on Graphics, August 2010.

[YPG01] Hector Yee, Sumanita Pattanaik, and Donald P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. ACM Trans. Graph., 20:39-65, January 2001.

Multi-View Random Fields and Street-Side Imagery

Michal Recky
ICG TUGraz
Graz University of Technology
Inffeldgasse 16
A-8010 Graz, Austria
recky@icg.tugraz.at

Franz Leberl
ICG TUGraz
Graz University of Technology
Inffeldgasse 16
A-8010 Graz, Austria
leberl@icg.tugraz.at

Andrej Ferko
FMFI UK
Mlynská dolina
842 48 Bratislava, Slovakia
ferko@sccg.sk

ABSTRACT

In this paper, we present a method that introduces graphical models into a multi-view scenario. We focus on a popular Random Fields concept that many researchers use to describe context in a single image and introduce a new model that can transfer context directly between matched images – Multi-View Random Fields. This method allows sharing not only visual information between images, but also contextual information for the purpose of object recognition and classification. We describe the mathematical model for this method as well as present the application for a domain of street-side image datasets. In this application, the detection of façade elements has improved by up to 20% using Multi-view Random Fields.

Keywords

Random Fields, Context, Redundancy, Image Stacks

1. INTRODUCTION

In a current computer vision research input data is often represented as large, redundant datasets with hundreds or even thousands of overlapping images. As the volume and complexity of data increases, it is no longer meaningful to employ manual inputs in any step of the process. This constraint on the work automation leads to a need to utilize as much information from images as possible. One potential approach is to employ "context". Most popular methods of context application are graphical models, specifically Random Fields. However, general Random Fields models are defined such that they allow observations only from a single image. This approach is limiting context as a feature of a single image, but the context is derived from objects in a real scene, from which an image is only one projection. How is this limiting context application and how can we expand the Random Fields model to cope with the presence of multi-view dataset is the topic of this paper.

The basic element in a Random Field model is a "site". This is generally a patch of image area ranging from a single pixel to a larger segment. In our application in a street-side images domain, a site is a rectangular area (block) of a building façade (see Figure 1). Each site has to be labeled according to visual data and a context in which it is observed. Context is defined as relations (spatial relations, similarity…) between sites. In a multi-view scenario, we have multiple matched images, each with its own set of sites. Extension of Random Fields into a multi-view is not straightforward, as the two sets of sites from matched images are typically overlapping.

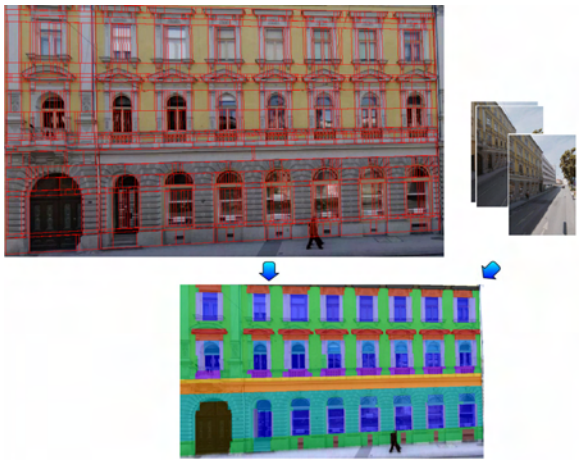


Figure 1: The application of Multi-View Random Fields for labeling of the façade elements. Top left – set of blocks that divide building façade into a set of sites for a graphical model. Bottom – final labeling is achieved as a combination of information from multiple overlapping images (for color-coding, see Figure 7).

Simple merging of these two sets would cause double detections of same objects and unresolved relations between sites. To solve both problems, we introduce a new concept – Multi-View Random Fields.

In this paper, the "*Background"* and "*Graphical Models"* sections are outlining a context of our work in a computer vision community and in a Random Fields models research. The "*Context in Multi-View"* section explains what type of context is relevant in multi-view and how it can be utilized. In the "*Multi-View Random Fields"* section the new graphical model is introduced and the *"Application of MVRF"* section present the illustrational application of the model in a street-side images domain.

2. BACKGROUND

The last decade saw growing interest in multi-view methods. With the introduction of a new generation of high resolution digital cameras and with rapid improvements in storage and computing hardware, multi-view imagery advanced from a source for the computation of point clouds by two-image stereo methods to a broad range of vision problems employing thousands of overlapping images. Open online image hosting sites (Flickr, Picasa, Photobucket…) have added interesting vision opportunities. While the basic principles for matching images remain applicable to such datasets [Har04a] [Leo00a], new problems needed to get solved, such as the organization and alignment of images without any knowledge about camera poses [Sna06a]. The resulting resource need in computing gets addressed by means of graphical processing units GPUs, or with distributed approaches [Fra10a]. Therefore current computer vision can cope with this avalanche of imagery and multi-views are becoming a common reality.

Extending the concept of Random Fields into such multi-view scenario comes from an idea that given more images of the same scene, more contextual relations can be examined. In this work, we present a mathematical model for Multi-View Random Fields that allows transferring contextual relations between matched images. We also present the application of Multi-View Random Fields in a domain of street-side images. This domain is useful for a demonstration, as there are large datasets of matched street-side images for the purpose of urban modeling (virtual cities, GIS, cultural heritage reconstruction) that establish a multi-view scenario. Urban scenes also exhibit strong contextual relations, as man-made objects adhere to an inherent organization. We show how façade elements can be classified, using both context and multi-view principles in one model.

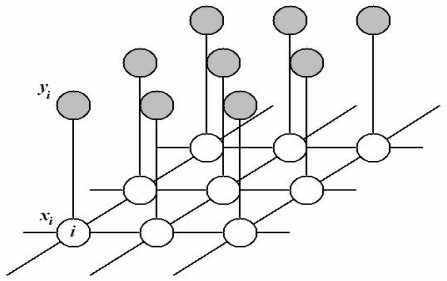


Figure 2. The typical application of MRF in computer vision. At each node (site) *i*, the observed data is denoted as y_i and the corresponding label as x_i. For each node, only local observations are possible. Generally each node represents a pixel in an image and observed data pixel's features.

3. GRAPHICAL MODELS

The most common non-causal graphical models in computer vision are Markov Random Fields (MRF).

MRF have been used extensively in labeling problems for classification tasks in computer vision [Vac11a] and for image synthesis problems. In a labeling task, MRF are considered to be probabilistic functions of observed data in measured sites of the image and labels assigned to each site. Given the observed data $\mathbf{y} = \{\mathbf{y}_i\}_{i \in S}$ from the image, and corresponding labels $\mathbf{x} = \{x_i\}_{i \in S}$, where S is the set of sites, the posterior distribution over labels for MRF can be written as:

$$P(\mathbf{x} \mid \mathbf{y}) = \frac{1}{Z_m} \exp\left(\sum_{i \in S} \log p(\mathbf{y}_i \mid x_i) + \sum_{i \in S} \sum_{j \in N_i} \beta_m x_i x_j \right), \quad (1)$$

where Z_m is the normalizing constant, β_m is the interaction parameter of the MRF and N_i is the set of neighbors of a site i. The pairwise term $\beta_m x_i x_j$ in MRF can be seen as a smoothing factor. Notice that the pairwise term in MRF uses only labels as variables, but not the observed data from an image. In this arrangement, the context in a form of MRF is limited to be a function of labels, thus allowing for semantic context (context between classes) and limiting geometric context to a structure of MRF graph (see Figure 2). This makes the MRF applicable mainly for simpler forms of local context.

To cope with such limitations, the concept of Conditional Random Fields (CRF) was proposed by J. Lafferty [Laf01a] for the segmentation and labeling of text sequences. The CRF are discriminative models that represent the conditional distribution over labels. Using the Hammersley-Clifford theorem [Ham71a], assuming only pairwise cliques potentials to be nonzero, the conditional distribution in CRF over all labels **x** given the observation **y** can be written as

$$P(\mathbf{x} \mid \mathbf{y}) = \frac{1}{Z} \exp\left(\sum_{i \in S} A_i(x_i, \mathbf{y}) + \sum_{i \in s} \sum_{j \in N_i} I_{ij}(x_i, x_j, \mathbf{y}) \right), \quad (2)$$

where Z is the normalizing constant, $-A_i$ is the unary and $-I_{ij}$ pairwise potential. The two principal differences between conditional model (2) and MRF distribution (1) are that the unary potential $A_i(x_i, \mathbf{y})$ is a function of all observations instead of only one observation $\mathbf{y}_i$ in a specific site i and the pairwise potential in (2) is also the function of observation, not only labels as in MRF. In CRF, the unary potential $A_i(x_i, \mathbf{y})$ is considered to be a measure of how likely a site i will take label x_i given the observation in a image $\mathbf{y}$. The pairwise term is considered to be a measure of how the labels at neighboring sites i and j should interact given the observed image $\mathbf{y}$. This concept of CRF allows for use of more complex context derived from larger sets of observations in the image and employing geometric context (e.g. spatial relations between objects). It is extended even more in a concept of Discriminative Random Fields [Kum06a], where an arbitrary discriminative classifier can be applied in a form of unary/pairwise potential.

However, in all concepts of Random Fields, the set of sites S (and thus the observations) is limited to a single image. How to extend these models into a multi-view is explained in subsequent sections.

4. CONTEXT IN MULTI-VIEW

Before the definition of a new Random Field model in multi-view, we must consider what type of context can be transferred between images. The most common type of context applied for classification is a local pixel context. In general, a small neighborhood around an examined pixel is taken as a context area and a graph structure of a model is placed in this neighborhood (one node per pixel). However, this approach is not suitable for multi-views, as neighborhoods around matched pixels in two images are in general uniform and will not present much useful additional information. Alternatively we can consider global context, which examines relationships between all objects in the images. In this type of context, we can observe different relations in different images, thus transferring such context would provide additional information for recognition and classification (see Figure 3). If spatial relations between objects are examined in this manner, graphical models are approximating spatial relations between objects in a real 3D scene.

In a standard Random Fields (RF) model, each image is considered a unique unit of information. Thus, we can consider a global context to be a specific feature of each image - the global context is a set of relations between all sites detected in a single image.

Figure 3. Building façade projected in slightly different views. Red lines (graph edges) represent spatial relationships between objects detected in the images, indicating different context in two projections for the same objects. For better overview, only some relations are visualized.

Typically, sites are either *pixels* or *segments*. Construction of a global model with node in each pixel would significantly increase the complexity of computation; therefore we consider segments as the representation of sites in our model.

Subsequently a site is represented by a specific area (segment) in a digital image. Such area represents an object (or part of object) and areas from two sites are not overlapping. In a general RF model, a set of all sites in one graph is denoted as S. In a local model, one set S include sites from a small patch of the image, however in a global model, S includes all sites from the entire image. Visual features of the area assigned to a specific site are denoted as image observation $\boldsymbol{y}_s$ from site $s \in S$. In a graphical model, if there is an edge between nodes assigned to sites s_1 and s_2, let's denote this relation as $\Phi(s_1, s_2) = 1$ and consequently if there is no edge between s_1 and s_2, denote this as $\Phi(s_1, s_2) = 0$.

Transferable Sites

Consider one image from the dataset as "examined image" to which we would like to transfer context from other matched images. Let's call any site $s \in S$ from an examined image a "native site". If the image matching is established in a dataset (we have a set of corresponding points that link images), we can look for any sites from other images that are *corresponding* to native sites. In most cases, sparse point cloud of matched points is enough to establish correspondence between site. Relative poses between images and camera parameters are not required. Definition of corresponding sites can vary in different applications. In general, *corresponding sites are two sites from different images that share some subset of corresponding points*;

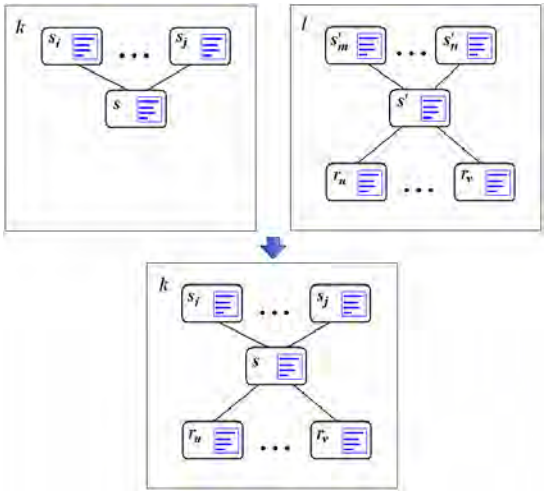


Figure 4. Transfer of sites from the image $l \in$ I to the image $k \in$I, as presented in Definition 1. Only sites from l that are not corresponding to any sites from k are transferred. This figure demonstrates only transfer between two images.

each site from matched images can have only one corresponding site in the examined image – the example of this relation is provided in the application section of this paper.

Given that corresponding sites usually represent the same objects, transferring such information between images would be redundant. Therefore we transfer sites that have no correspondences in the examined images to provide new information. We denote such sites as "transferable sites". For a single, examined image from the image stack, let's define the set of transferable sites as:

Definition 1: *If* $S_k = \{s_1, s_2, \ldots, s_n\}$ *is the set of sites for single image* $k \in$ I, *where* I *is the set of images and correspondences have been established between the images from* I *such that* $s_i' \in S_l$ *is a site from image* $l \in I-\{k\}$ *corresponding to a site* s_i. *Than the* $R_k = \{r_1, r_2, \ldots, r_m\}$ *is the set of transferable sites for the image* k *if* $\forall r_j \in R_k \exists s_i \in S_k \mid \Phi(r_j, s_i') = 1$ *and* $\forall r_j \in R_k \neg \exists r_j' \in S_k$. R_k *is constructed such that* $\forall r_i, r_j \in R_k$, r_i *and* r_j *are not correspondent to each other in any two images from I*

Thus the R_k is the set of sites from other images than k, that are in the relationship in graphical model with some corresponding site to sites from S_k, but themselves have no correspondences in S_k (see Figure 4). The set of transferable sites can be seen as a context information, that is available in the image stack, but not in the examined image. If sites are the representations of objects, than in a transferable set, there are objects in context with the scene of the image that are currently not located in the projection, thus are occluded, out of the view or in different timeframe. This also means that the visual information from the sites in R_k are not present in the image k. If the sites from R_k are included in the vision process, they can provide additional context and visual information that is not originally present in the examined image.

Note that a transferable site is not equivalent to a native site in an examined image. Even though transferable sites have the same set of visual features as sites native to the image and they can be assigned the same set of spatial and contextual relations in a graphical model, transferable sites lost all original contextual relationships except the relationships to the sites they are connected within the examined image. This makes them harder to label. But the labeling of transferable sites is not the aim in the case of examined image (the goal is to label only native sites), thus transferable sites can contribute information for image labeling, but the labeling of themselves is usually irrelevant.

5. MULTI-VIEW RANDOM FIELDS

Given a non-equality of transferable sites to native sites, standard RF models are not compatible with this extended set. For this reason, we introduce a new model denoted as *Multi-View Random Fields* (MVRF). This model is derived from a CRF, described in Section 2; however we extend the posterior probability distribution into MVRF model framework as follows:

Given the observed data $\mathbf{y} = \{\mathbf{y}_i\}_{i \in S}$ from the image, corresponding labels $\mathbf{x} = \{x_i\}_{i \in S}$, where S is the set of native sites from the image and observations from transferable set $\mathbf{z} = \{\mathbf{z}_i\}_{i \in R}$ with corresponding labels $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i \in R}$, where R is the set of transferable sites, the posterior distribution over labels is defined as:

$$P(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) = \frac{1}{Z} \exp\left(\sum_{i \in S} A_i(x_i, \mathbf{y}) + \sum_{i \in R} A_i'(\tilde{x}_i, \mathbf{z}_i) + \right.$$
$$\left. + \sum_{i \in S} \left(\sum_{j \in N_i} I_{ij}(x_i, x_j, \mathbf{y}) + \sum_{j \in K_i} I_{ij}'(x_i, \tilde{x}_j, \mathbf{y}, \mathbf{z}_j) \right) \right) \quad ,(3)$$

where Z is the normalizing constant, N_i is the set of native sites neighboring site i and K_i is the set of transferable sites neighboring site i. - A_i and - A_i' are unary potentials, - I_{ij} and - I_{ij}' are pairwise potentials (for native sites and transferable sites respectively). The differences between potentials for transferable sites and for native sites are as follows:

- In the unary potential for a transferable site, only observations from the site itself are

considered, instead of observation from the entire image for native sites. This is due to the fact, that a transferable site does not have any connections to the image except for the site it is neighboring. Even if other connections exist (with other sites in the image), it is a hard task to establish relationships. For native site, there are no changes to a standard conditional model.

- In the pairwise potential, in addition to observation from the image, local observation from the transferable site is considered, when relations are examined between a native site and transferable site. The inclusion of all image observation grant at least the same level of information in pairwise computation as in a standard CRF model and the additional observation form transferable site represent extended context for native image observation. The pairwise potential for two native sites is the same as in a standard CRF model.

This model has some additional unique characteristics. For example, no pairwise relations are considered between two transferable sites. This is based on the construction of transferable sites set. A site from such set can be neighboring several native sites, but not any other transferable site. This can be seen as a limitation for the model, however without additional high frequency information about the scene (as a prior knowledge), it is virtually impossible to establish relationships for transferable sites.

The computational complexity of the model is not increased significantly. Pairwise potentials are computed only for native sites, as it is in the standard CRF model. The difference is in the number of neighbors for each site, however even this number should not increase significantly. When considering a global model, each new neighbor (transferable site in relation to the native site) represents a new object in the projection. This is dependent on the differences between projection parameters – camera positions, optical axes…, but even for very different parameters, the number of objects should not differ significantly for the same scene. From the general observation, the number of neighboring transferable sites is notably lower than the number of neighboring native sites.

Potentials Modifications

Unary potential for native image sites, similar to a standard CRF is a measure of how likely a site i will take label x_i given the observations in image $\mathbf{y}$. A standard approach described in a work of S. Kumar is to apply Generalized Linear Models (GLM) as local class conditional [Kum06]. In that case, given the model parameter $\mathbf{w}$ and a transformed feature vector at each site $\boldsymbol{h}_i(\mathbf{y})$, the unary potential can be written as:

$$A_i(x_i, \mathbf{y}) = \boldsymbol{log}\left(\sigma\left(x_i \mathbf{w}^T \boldsymbol{h}_i(\mathbf{y})\right)\right) \quad ,(4)$$

For the transferable sites, the feature vector is limited to the observations from single site. This limitation defines a new expression for unary potential, exclusive to transferable sites as

$$A_i'(\tilde{x}_i, \mathbf{z}_i) = \boldsymbol{log}\left(\sigma\left(\tilde{x}_i \mathbf{w}^T \boldsymbol{h}_i(\mathbf{z}_i)\right)\right) \quad ,(5)$$

The feature vector $\boldsymbol{h}_i(\mathbf{z}_i)$ at the transferable site i is defined as a nonlinear mapping of site feature vectors into high dimensional space. The model parameter $\mathbf{w} = \{w_0, \mathbf{w}_1\}$ is composed of bias parameter w_0 and model vector $\mathbf{w}_1$. $\sigma(.)$ is a local class conditional, that can be any probabilistic discriminative classifier.

The pairwise potential for two native sites from the image remains the same as in CRF model, given the GLM are applied to compute the class conditional:

$$I_{ij}(x_i, x_j, \mathbf{y}) = \beta\left(K x_i x_j + (1-K)\left(2\sigma\left(x_i x_j \mathbf{v}^T \boldsymbol{\mu}_{ij}(\mathbf{y})\right) - 1\right)\right) \quad ,(6)$$

where $0 \le K \le 1$, $\mathbf{v}$ and β are the model parameters and $\boldsymbol{\mu}_{ij}(\mathbf{y})$ is a feature vector. For transferable sites, we introduce the additional feature vector in a form of observations from specific site:

$$I_{ij}'(x_i, \tilde{x}_j, \mathbf{y}, \mathbf{z}_j) = \beta\left(K x_i \tilde{x}_j + (1-K)\left(2\sigma\left(x_i \tilde{x}_j \mathbf{v}^T \boldsymbol{\mu}_{ij}(\mathbf{y}, \mathbf{z}_j)\right) - 1\right)\right) \quad ,(7)$$

where $\mu_{ij}(\mathbf{y},\mathbf{z}_i)$ is a feature vector defined in a domain $\mu : \Re^{\gamma} \times \Re^{\gamma} \to \Re^{q}$ such that observations are mapped from the image/sites related to site s into a feature vector with dimension γ. Note that the smoothing term $K x_i \tilde{x}_j$ is the same as in a standard CRF definition. Thus if $K = 1$, the pairwise potential still performs the same function as in a MRF model, however given new transferable sites, the smoothing function will depend also on their classification $\tilde{x}_j$. In this case, visual information from transferable sites is not involved in the pairwise term and is only applied in the unary term. If $K<1$ the data-dependent term $2\sigma\left(x_i \tilde{x}_j \mathbf{v}^T \boldsymbol{\mu}_{ij}(\mathbf{y}, \mathbf{z}_j)\right) - 1$ is included in a pairwise potential. Observations from the image related to the examined native site and observation from transferable site are transformed into feature vector and involved in computation.

Parameter Learning and Inference

In this work, we constructed an MVRF model to be as compatible with other RF models as possible. This approach is observed also in a parameter learning process, as any standard method used for learning of

CRF model can be also used for MRVF model. To further simplify the process, we observed that learning from single (un-matched) images is feasible without the loss of strength of the model. This is due to the construction of potentials - in a unary potential, visual features do not change for transferable sites, therefore they can be learned directly from single images in training dataset. The spatial relations defined for a pairwise potential also do not change significantly for the pair native-transferable site. For such reasons, we can assume that the MVRF model can be learned even directly from single images without dataset matching. Therefore, methods such as pseudo-likelihood can be applied for learning.

Similarly, parameter inference can be performed, using any standard method applied in CRF. In our application, we use Belief Propagation, but other possible methods are Tree-Based Reparameterization or Expectation Propagation for example.

6. APPLICATION OF MVRF

In this section we present the application of MVRF in the building façades dataset for the purpose of façade elements detection and classification. This application is based on the dataset provided by a vehicle-based urban mapping platform. Sparse image matching is applied (see Figure 5), using the Structure-from-Motion method [Irs07a]. We selected the left camera subset, since it provides a clear view of the building façades, not distorted by the perspective (which, however, is easy to rectify) and with good visual cues. This setting will demonstrate the advantages of MVRF in cases when a site was misdetected and presents lost contextual information in standard models. In most images, the building façade is not projected in its entirety and parts are located in other images. Therefore in such cases, the MVRF will also provide new contextual and visual information in a form of transferable sites based on the objects that are not located in the original image.

In each image, separate facades are detected. This can be achieved when the wire-frame models of the scene are available, or using visual cues, such as repetitive patterns [Rec11a]. Subsequently, a modified gradient projection is applied to segment each façade into a set of blocks. This method is based on a standard gradient projection approach [Lee04a] designed for the detection of windows with following modifications:

First, we vertically project gradients to establish a horizontal division of the façade into levels (level borders are located at the spikes of the projection). Subsequently, we compute horizontal gradient projections in each level separately.

Figure 5. Top row: two examples of the same façade, matched with a sparse point cloud (red dots). Middle row: set of blocks located in each façade (left image show façade detail for better overview, right image entire facade). Bottom row: set of blocks from the first image projected into a second image and a set of transferable sites (highlighted blocks) that is derived from the projection (as sites that have no correspondence in second set).

This process will yield a set of blocks bordered by level borders horizontally and spikes in projection vertically (see Figure 5). Second, we consider each block as a site for a graphical model, thus we compute visual features for each block and consider spatial relationships between blocks. Visual features, such as texture covariance, or clustering in a color space are used for classification [Rec10a]. For example, clusters in a CIE-Lab color space are computed for each block and are compared to class descriptors.

When the segmentation of a façade into a set of block is established, we can define a global graphical model in this structure. Each block is considered a site, thus each node of the graph is placed in a separate block. We define neighborhood relation such that for each block, its neighbors are all blocks located in areas above, below, left and right from itself (see Figure 6). This definition allows considering all objects at the same level and column to be involved in contextual relations, accounting for relations, such as rows and columns of windows, or window-arch. An edge of a graphical model is placed between each two neighboring blocks. In this approach, a separate graph is created for each façade in the image.

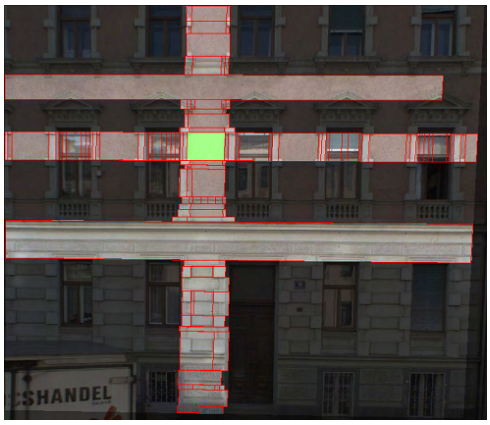


Figure 6. Example of site neighborhood, as defined in this application. Green block is the examined site and highlighted blocks are defined as its neighborhood.

Multi-View Scenario

To establish a multi-view, we use a sparse point cloud. We match blocks between images such that we interpolate between detected corresponding points to achieve rough point-to-point matching. If two blocks in different images share at least 2/3 of matched points (detected and interpolated), we define these as corresponding blocks. Given one image as "examined", we can label all blocks from the same façade in other images as either corresponding or non-corresponding. Subsequently, transferable sites are blocks that are from the same façade as in an examined image, but are non-corresponding to any block from the examined set (see Figure 5). Establishing the relations between native and transferable sites is straightforward, as we can still consider up, down, left, right directions. With these definitions, we can construct the MVRF model from our dataset.

Experiments

We use the described model for the purpose of façade elements detection and classification. The set of classes with corresponding color coding is displayed in Figure 7. Our testing dataset consists of 44 matched images. This dataset covers three full building façades and one half façade. A sparse point cloud of 1429 3D points is used to match images. Approximately 800 points are projected into each image. In the testing process, we compare the number of façade elements to the number of detected elements with the applied method. We counted overall numbers of elements through the entire dataset, as displayed in Table 1. For example, total number of 536 "window centre" elements can be observed in all images, that is approximately 12 "window centers" per image.

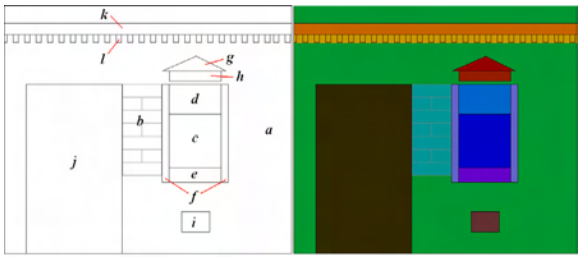


Figure 7. Set of classes: *a*) clear façade; *b*) brick façade; *c*) window centre; *d*) window top; *e*) window bottom; *f*) window margins; *g*) arch top; *h*) arch bottom; *i*) basement window; *j*) door; *k*) ledge; *l*) ledge ornament; On the right side, color representation of each class is displayed.

Each façade was processed separately, that is if there were two façades in one image, such image was processed two times (each time for different façade). After running the algorithm, a number of detected elements is counted visually. The façade element is defined as *detected*, if at least 2/3 of its area is labeled with the corresponding class. For the training purpose, we used the subset of 3 images from the dataset and other 5 unrelated images as labeled ground truth. This proved to be sufficient, as the spatial relations between classes are in general stable through different facades and a certain visual features variability

Class	# el	single	multi /native	multi /trans
clear façade	61	61	61	61
brick façade	54	54	54	54
win. centre	536	485	531	531
window top	311	270	303	308
win. bottom	300	227	273	288
win. margin	683	572	618	654
Arch top	199	176	189	192
Arch bottom	199	184	194	194
Basem. win	121	98	115	117
Door	34	32	33	33
Ledge	90	90	90	90
Ledge orna.	34	32	34	34

Table 1. The Results for the MVRF application. "# el" displays the overall number of each class for entire dataset (44 images). "single" displays detected elements in MVRF single image scenario (equivalent to CRF), "multi/native" displays results for multi-view scenario with only native sites in results and "multi/trans" display results for multi-view scenario with transferable sites labels in results. Numbers displayed are the detected façade elements in all images of dataset.

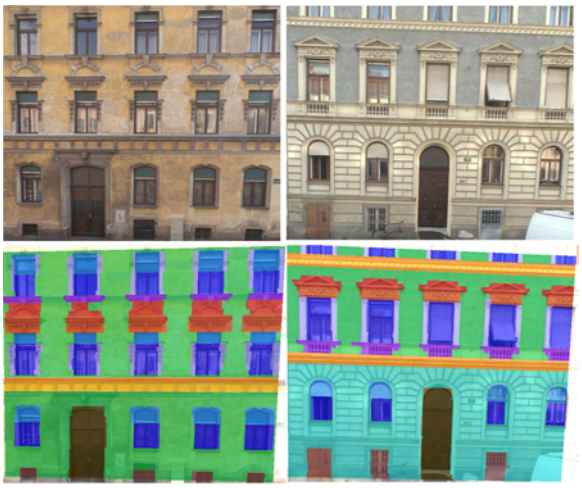


Figure 8. Two examples of classification results. Classes are labeled according to color scheme explained in Figure 7. Colors are superimposed over original images in the bottom row.

was allowed by the use of descriptors (e.g. clustering). We trained on single images without the use of matching. For the parameter inference, we used a Belief Propagation method. Initial classification was performed based on only visual features and in each iterative step of the method, it was refined by pairwise relations and site features described in a model. In each step, we also refined visual descriptors for each class to better approximate features in each unique façade. Results can be observed in the Table 1. We included results for scenarios, where no transferable sites were used (single), and the MVRF model is equivalent to CRF in this case, results when only labels of native sites were considered and results were labels of transferable sites were included. Notice a significant improvement in detection for classes that are visually ambiguous, but have strong contextual relations (e.g. window margins, window tops). For a "win. bottom" class, the correct detection rate improved from 76% in a single-view to a 96% in a multi-view with transferable sites projected, thus achieving a 20% improvement. Results illustrated in Figure 8.

7. CONCLUSION

In this paper, we addressed a common problem in a current research – how to work with context information in matched datasets and to alleviate an artificial limitation of graphical models to single images. We introduced a new MVRF model directly applicable in a multi-view scenario. We extended the standard CRF model such that it can work with overall context of the scene present in the multi-view dataset, but it still retains the same properties for processing visual and contextual information in a single image. Validity of this model is subsequently demonstrated in the application in street-side image domain – detection of façade elements. However the new MVRF model is applicable in same situations as a standard CRF model, provided that appropriate image matching is available. For example, the MVRF model was also used for a super-pixel based semantic segmentation of outdoor images in our other work.

8. REFERENCES

[Fra10a] Frahm, J. M., et al. Building Rome on a Cloudless Day. *European Conference on Computer Vision,* pp. 368–381, 2010

[Ham71a] Hammersley, J. M., Clifford, P. *Markov* field on finite graph and lattices. Unpublished, 1971

[Har04a] Hartley, R. and Zisserman, A. Multiple View Geometry in Computer Vision. *Cambridge University Press*, ISBN: 0521540518, 2004

[Irs07a] Irschara, A., et al. Towards wiki-based dense city modeling. *International Conference on Computer Vision*, pp. 1-8, 2007

[Kum06a] Kumar, S. and Herbert, M. Discriminative random fields. *International Journal of Computer Vision*, 68(2), pp. 179–201, 2006

[Laf01a] Lafferty, J., et al. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning*, pp. 282-289, 2001

[Lee04a] Lee, S. C. and Nevatia, R. Extraction and integration of window in a 3d building model from ground view images. *Computer Vision and Pattern Recognition*, pp. 112-120, 2004

[Leo00a] Leonardis, A., et al. Confluence of computer vision and computer graphics. NATO science series, *Kluwer Academic Publishers*, ISBN 0-7923-6612-3, 2000

[Rec11a] Recky, M,. et al. Façade Segmentation in a Multi-View Scenario. *International Symposium on 3D Data Processing, Visualization and Transmission*, pp. 358-365, 2011

[Rec10a] Recky, M. and Leberl, F. Windows Detection Using K-means in CIE-Lab Color Space. *Internation Conference on Pattern Recognition,* pp. 356-360, 2010

[Sna06a] Snavely, N., et al. Photo tourism: Exploring photo collections in 3d. *ACM Transactions on Graphics*, pp. 835 – 846, 2006

[Vac11a] Vacha, P., et al. Colour and rotation invariant textural features based on Markov random fields. *Physical Review Letters, No. 6*, pp. 771-779, 2011

Collision Detection on Point Clouds Using a 2.5+D Image-Based Approach

Rafael Kuffner dos Anjos
Inesc-ID
Av. Prof. Dr Anibal Cavaco Silva
Portugal 2744-016, Porto Salvo
rafaelkuffner@gmail.com

João Madeiras Pereira
IST/Inesc-ID
Rua Alves Redol, 9
Portugal 1000-029, Lisboa,
jap@inesc-id.pt

João Fradinho Oliveira
C3i/Inst. Politécnico de Portalegre
Praça do Município
Portugal 7300, Portalegre
joaofradinhooliveira@gmail.com

ABSTRACT

This work explores an alternative approach to the problem of collision detection using images instead of geometry to represent complex polygonal environments and buildings derived from laser scan data, used in an interactive navigation scenario. In a preprocessing step, models that are not point clouds, are sampled to create representative point clouds. Our algorithm then creates several 2.5+D maps in a given volume that stacked together form a 3D section of the world. We show that our new representation allows for realistic and fast collision queries with complex geometry such as stairs and that the algorithm is insensitive to the size of the input point cloud at run-time.

Keywords

Collision Detection, Point-based Graphics, Navigation

1. INTRODUCTION

Collision detection is normally a bottleneck in the visualization and interaction process, as collisions need to be checked at each frame. Traditionally, the more complicated and crowded is our scene, the more calculations need to be done, bringing the frame-rate down. Therefore the optimization of this process, gaining speed without losing quality in the simulation, is something that has been researched for years. Although several different techniques and approaches have been developed, and showed good performance in specific scenarios, these approaches rely on object topology information which is easily extracted from polygonal models. However with point cloud models, the classical approaches either will not work, or will have to heavily adapt to this specific scenario, compromising their optimizations.

Using images as an alternative way of executing the task of collision detection might just be the answer. Image-based techniques can have their precision easily controlled by the resolution of the used images, and the algorithms are completely independent of the object's topology and complexity at run-time. It does not matter whether an object has sharp features, round parts, or even whether it is a point cloud, as all we are dealing with is the object's image representation. Being a scalable and promising technique, Image-based collision detection seems to be a plausible alternative to the classical approaches.

Our approach focuses in a virtual reality navigation scenario, where the scene is derived from the real world via devices such as laser scanners, which tend to generate enormous point clouds. Also, the hardware at hand might not fit the basic requirements for most algorithms and techniques, a situation that commonly will happen in tourism hotspots, museums, or other places where we would like ordinary people to be able to interact with the system. The developed application enables them to control an avatar on a static environment, a point cloud or polygonal model.

The main contribution of our research is a new 3D world representation for environment and buildings which is completely image-based with information that enables realistic and robust collision detection with underlying complex geometry such as slopes and stairs. Slicing a given structure along the Z axis (Using Cartesian coordinates as illustrated in Figure 1); we create a discrete set of images containing height information about the surface, and possible collidable frontiers. It is a flexible format that is able to represent either point clouds or polygonal models. This representation allows us to perform collision detection with user chosen precision, and high scalability.

2. RELATED WORK

The problem of collision detection is present in several areas of research and applications, each of them having different concerns, requirements and desired results. This necessity has prompted the creation of several techniques that try to deliver these results using varied approaches, each one fitting to specific problems.

Feature based: Famous examples are the Lin-Canny algorithm [MLJC] and its more recent related algorithm, V-Clip [BMM], which keeps track of the closest features between two objects, deriving both the separation distance, and the vertices that have possibly already penetrated the other object.

Bounding Volume Hierarchies: Different volumes have been developed to envelop the basic features, such as Spheres [PMH], Oriented Bounding Boxes (OBB) [SGMCLDM], or Axis Aligned Bounding boxes (AABB) [TLTAM] [MJBJ] [XZYJK], each of them has its advantages over the others; Spheres are easier to fit, OBBs have faster pruning capabilities, and AABBs are quicker to update, therefore being very popular in deformable body simulation.

Also, different tree traversing and creation techniques [TLTAM] [SGMCLDM] have been developed to optimize these expensive operations, taking into account each specific kind of application.

Stochastic algorithms: Techniques that try to give a faster but less exact answer have been developed, giving the developer the option to trade accuracy in collisions with computing power. The technique based on randomly selected primitives, selects random pairs of features that are probable to collide, and calculates the distance between them. The local minima is kept for the next step and calculations are once again made. The exact collision pairs are derived with Lin-Canny [MLJC] feature based algorithm. With a similar idea, Kimmerle et. al [SKMNFF] have applied BVH's with lazy hierarchy updates and stochastic techniques to deformable objects and cloth, where not every bounding box is verified for collision, but it has a certain probability.

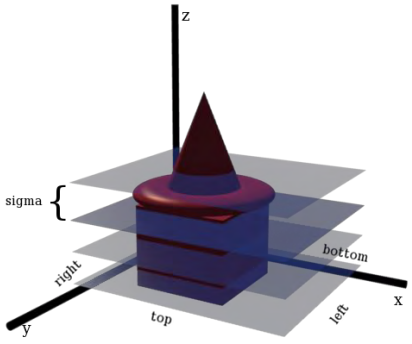


Figure 1. Slices creation process, and camera variables.

Image-based algorithms: These solutions commonly work with the projection of the objects, in contrast with previous techniques that work in object space, meaning that they are not dependent of the input structure, and as such are more suitable to point clouds. However to our knowledge they have not yet been applied to point clouds.

RECODE [GBWHS] and several other works [DKDP] [KMOOTK], [GBWSW] take advantage of the stencil buffer and perform collision detection on it by using object coordinates as masks, and thus detecting possible penetrations.

CULLIDE [NSRMLDM] uses occlusion queries only to detect potentially colliding objects, and then triangle intersection is made on the CPU. They render the bounding volumes of the objects in normal and reverse list storage order, and remove the objects that are fully visible in both passes, meaning that these objects are not involved in any collision.

Heidelberger et. al [BHMTMG] uses simple AABB's as bounding volumes for the objects in the scene. Potentially colliding objects are detected, and a LDI (Layered Depth Image [JSLR]) of the intersection volume between the two objects is created. That is, a volumetric representation of an object across a chosen axis. At each rendering step, as a polygon is projected into the LDI, the size of the intersubsection volume is computed. Faure et. al [FSJF] [JFHFCP] addresses not only collision detection, but also its response by using this same principle.

On a collision avoidance scenario, we want to predict an upcoming collision, and use this information to prevent it from happening. It is used mostly in artificial intelligence to control intelligent agents or robots. Loscos et. al [CLFTYC] represent the environment as patches to where the agent can or cannot go according to its occupation. It can be considered a basic but robust image based approach since it uses a 2D map to represent the environment.

Collision detection on point clouds: Algorithms using feature based techniques, bounding volumes, and spatial subdivision have been developed. Klein and Zachmann [JKGZ] create bounding volumes on groups of points so collision detection can be normally applied. Figueiredo et. al [MJBJ] uses spatial subdivision to group points in the same voxel, and BVHs to perform collision detection. The main issue while dealing with point clouds is the absence of closed surfaces and object boundaries. Ordinary BVH or stochastic techniques have to heavily adapt to this scenario, normally leading to not so efficient hierarchies. Feature-based techniques that work at vertex level are not scalable enough to be suited to these scenarios, since point clouds are normally massive data sets. Image-based techniques have the

```
for all points p in m do
    s ← floor((abs(z − z_min)) / σ)
    red ← (abs(z − z_min) mod σ) / σ
    red_old ← cube[x_screen][y_screen][s]
    if (abs(red_old − red)) / σ > σϵ then
        cube[x_screen][y_screen][s] ← 1
        p.color(red, 1, 1)
        if red_old > red then
            p.z ← p.z + (red_old − red) * σ + ϵ_2
        end if
    else
        cube[x_screen][y_screen][s] ← red
        p.color(red, 1, 0.1)
    end if
end for
```

Figure 2. Points Coloring and Obstacle Detection Algorithm

disadvantage of sometimes requiring special graphic card capabilities, but only for some implementations. Overall, their properties make them the best suited for the proposed scenario of the tourism kiosk.

For further information on collision detection and avoidance techniques we suggest the following surveys: [NMAS] [SKTGKICB] [MT].

3. IMPLEMENTATION

Image-based algorithms that have been presented in the community ([NSRMLDM] [GBWHS] [DKDP] [NBJM] [JFHFCP] [FSJF]) perform very well in various kinds of scenarios, but some features of our set scenario (described on Section 1) make them hard or impossible to be applied (e.g. our data is unstructured, not all objects are closed or convex).

Our work extends the idea of a 2.5D map presented on the work of Loscos et. al [CLFTYC] by using enhanced height-maps, where the pixel color not only represents the height on that point, but also contains obstacle information, while at the same time overcoming the limitations of only supporting single floor environments. We call these enhanced maps, 2.5+D maps. Instead of having just one height map, we create a series of enhanced maps along intervals sized σ on the z axis, thus enabling the storage of more than a single z value for each (x, y) pair. Unlike LDI's, our representation does not combine several images into a volumetric representation, but separates each slice into a single image so we can easily perform memory management, and apply image comparison and compression techniques to have a better performance. Using the color of each pixel as a representation of a voxel, we write height information on the red channel, and identify obstacles on the blue channel. By adding these variables, we can determine not only the height level where the avatar should be standing, but also if he is colliding with any obstacle in several different heights.

Slices Creation

The creation of this representation is executed in a pre-processing stage, which is composed of several steps that must be performed from the input of the model until the actual rendering to create the snapshots that will be the used as collision maps. These slices are created as following. The camera is first set up according to the previously calculated bounding boxes of the input model on an orthogonal projection. After rendering that projection, a snapshot sized σ is created and saved into the disk for further use. The camera then is moved along the z axis, and the process is repeated until the whole extension of the model has been rendered into images. A visual representation of the described variables along with the slices computed on an example model can be seen on Figure 1 and two real slices can be seen on Figure 4.

Polygonal Model Oversampling

We aim for a solution that accepts both polygonal models and point clouds. However these representations are inherently different and cannot be processed initially in the same way. Hence we created a solution that approximates the polygon models with a synthetic point cloud thus enabling later steps to be processed in the same way. We apply a simple oversampling operation that operates at triangle level transforming a polygonal model into a point cloud with a user-choice level of precision. After oversampling and discarding the polygons, the rendering of the produced point cloud has exactly the same shape and fill as if rendering the polygonal representation to the height map.

Information Processing and Encoding

Since all of our collision information will be written on collision maps as colors, we must assign each point on the point cloud a color representing its collision information. This will not replace the original color of that point in question. When writing these points on the output image, each pixel will represent a voxel sized (t, t, σ) on object space. So the painted color on that pixel will represent all the points contained in that volume. The algorithm on Figure 2 performs both the operation of height map information encoding, and obstacle detection. We define σ as $3t$, so as to ensure that one has more than one point on each slice, to properly perform the obstacle detection, as will be described later.

The first operation is executed as follows: We calculate the difference between the current point z coordinate and the model's lower bounding box z_{min}, and apply the modulo operator with σ. This remainder r represents the points z coordinate on an

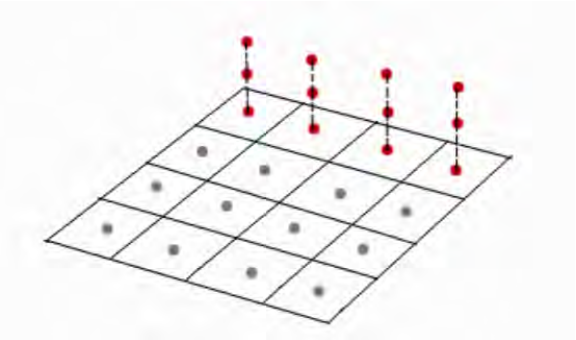


Figure 3. Technique for surface orientation detection. Red points belong to a vertical structure, grey points to the floor.

interval $[0, \sigma]$. To be used as a color value, this difference must belong to the interval [0,1], so we calculate r/σ thus deriving finally the red channel value. The simplified formula is given by $red = \frac{abs(z - z_{min}) mod\, \sigma}{\sigma}$

As navigation on a real-world scenario is normally performed horizontally on the xy plane, we classify an obstacle as a surface that is close to being perpendicular to xy, parallel to zy. So our obstacle collision technique simply estimates how parallel to the z axis a certain surface is. Figure 3 illustrates how potential obstacles and floor are classified using the coloring algorithm (Figure 2). Points lined up vertically on the same pixel most likely belong to a vertical surface. Diagonal structures like ramps are climbable up to a certain degree. The closer to a wall they are, the greater the probability is that its points are considered to be obstacles.

In order to keep track of point information that will be stored in the slices we use an auxiliary structure, a 3D array $cube[w][h][\sigma]$, after processing each point, we keep its color value on the position of the array representing the voxel on object space from where it came from. If there is already a stored value in this voxel, the difference between both red values is calculated, and transformed into an object-space distance $\frac{abs(red_{old} - red)}{\sigma}$.

If this difference is bigger than a certain small percentage ε (e.g. 7%) of the size σ of the slice, we assume that the points are vertically aligned, belonging to a vertical surface. These points are marked on their blue channel with the value 1, and we slightly increase its z coordinate so that the point is not occluded when rendering the maps. Typical output slices produced in the pre-processing stage can be seen in Figure 4, an office environment, where the floor has been correctly labeled as green, and all the walls are labeled as white or light blue.

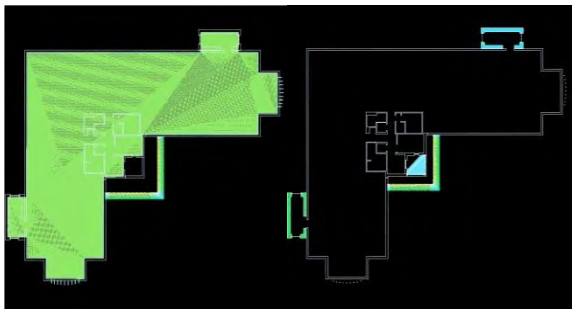


Figure 4. Two slices of an office environment, where walls (white/blue) and floor (green) are clearly distinguished, as well as a subsection of a roof (green to yellow) on the entrance.

Collision Detection

The developed representation provides us with enough information to perform quick collision detection on the navigation scenario given on section 1 where we consider point clouds as a viable input. In the accompanying video we show precise collision detection between rigid bodies and point clouds.

We divide the task of collision detection into two steps: a first step, that we call Broad phase, where we verify the occurrence of collisions between any objects in the scene, and a second step called narrow phase, where we perform collision response.

3.1.1 Broad Phase and Collision Detection

This task consists on identifying possible collisions between all objects on the scene. By representing the avatar that will be navigating on the environment by an Axis Aligned Bounding Box (AABB), we first calculate its size in pixels by calculating $pix_x = \frac{size_x}{t}$ and $pix_y = \frac{size_y}{t}$, where threshold t is calculated as the pixel size. This will be the number of pixels checked for collision on each slice, around the center of the avatar. If any checked pixel is not black, we mark the object as colliding, and they will be further processed in a narrow phase.

The only images we will need to load into the memory at the same time in order to perform collision detection between the given avatar and the environment are the ones located between $slice_0 = \frac{zp_{min} + zp - z_{min}}{\sigma}$ and $slice_n = \frac{zp_{max} + zp - z_{min}}{\sigma}$, where zp represents the z coordinate of the avatar. These slices contain collision detection information about the location where the pawn currently is.

New slices that are needed, are loaded into memory until a user defined constant of number of slices (n_{slices}) is reached. New slices beyond this point, replace an already loaded slice that has the furthest z value from the avatar's own z value, meaning it is not needed at this point of the execution. In practice we found that six slices covered well the avatar's

potential waist, shoulders, head, knees, and feet collisions with the scene.

3.1.2 Narrow Phase and Collision Response

In our current version, the sole purpose of our collision response algorithm was to simply avoid objects from overlapping, and provide a basic navigation experience on the given environment. Any other more complex technique could be applied here, but this simple solution fulfills the requirements for our navigation scenario. We focused on an efficient broad phase algorithm, and a flexible representation so we could apply any chosen image-based technique on the narrow phase. This was achieved with a straightforward extension of our broad-phase algorithm, by applying the concepts of collision response from height maps, and collision avoidance [CLFTYC]. Instead of simply returning true when we find pixels that are not black, we gather information for collision response each time we find colored pixels. Pixels with the blue channel set to 1 always represent an obstacle, except on applications where we want to enable the avatar to climb small obstacles, as the agents from Loscos et.al [CLFTYC]. On these situations, we may ignore these pixels up until the height we want to be considered as climbable. As our avatar moves on fixed length steps, and each time it collides we correct it to the place it was on the previous check, we thus ensure that the avatar is always on a valid position. We apply this (x, y) correction each time an obstacle pixel is found until all the pixels representing the avatar's bounding box are verified.

Height is defined exactly as it is when precomputing height maps. By multiplying the coded height information on the red channel by σ and adding the z base coordinate of the given slice, we have precise information about a given point's height. Collision response can be made by setting the final height to the average height of the points on the base of the bounding box, or by the adopted strategy, the maximum value. Here we also check for surface height values from the first slice until the height we want to consider as climbable.

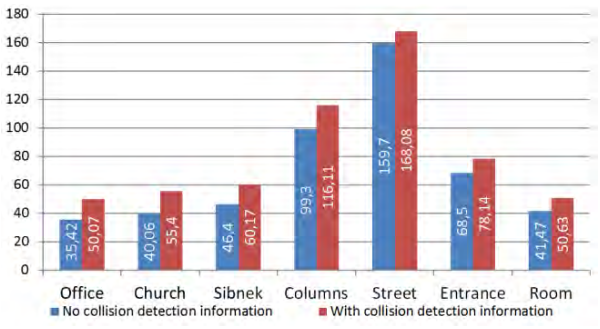


Figure 5. Memory load at a given moment during runtime on a 700x700 configuration, 6 slices, with and without collision detection.

The complexity of this operation is exactly $O(pix_x * pix_y * s)$ where pix_x and pix_y are the number of pixels ocuppied by the base of the avatar and s is the number of slices encompassed by the avatar's height. We point however, that these checks are already performed in the broad phase, and hence can be re-used in the narrow phase without adding any extra operations.

4. EXPERIMENTAL RESULTS

We have implemented the whole algorithm using OpenGL 2.1.2, C and the OpenGL Utility Toolkit

(GLUT) to deal with user input and the base application loop management. The platform used for tests was a computer with an Intel core 2 Duo CPU at 2 GHz with 2GB of RAM, a NVIDIA GeForce 9400 adapter, running Microsoft Windows Seven x86.

Table 2 shows the time taken on the whole preprocessing stage for each model and configuration. Polygon sampling during the preprocessing of polygonal models is clearly the most computationally intensive task in the pipeline, as the two higher complexity point cloud models (Entrance and Room as seen on Table 1 and Figure 7) that did not require sampling had a faster performance. In the preprocessing phase the increase on processing time with point clouds is linear to point complexity. This linear growth is expected since each point must be checked for coloring once, and also for common input processing tasks such as file reading and display list creation.

Regarding the results of the overall memory cost specifically in the preprocessing phase, Table 2 shows that memory scales almost linearly according to the input size of the point cloud (Entrance versus Room in Table 1). This memory is needed temporarily for allocating the auxiliary 3D array for obstacle detection in the slice creation phase. Similarly, tests have shown that this memory consumption also grows linearly with the number of points produced in the polygon sampling phase.

During the application runtime, the average memory consumption varies according to the number of loaded slices into RAM, and according to the size of the loaded model used for rendering (Figure 5 and Table 1). On a 700x700 resolution, the peak minimal value found in any model we experimented was 50,07MB (Office) and the peak maximum 168,08MB (Street), with 6 slices loaded in memory. Table 2 shows how much memory the application consumes when only rendering the models and the total memory with collision detection, while having 6 slices loaded in memory. By controlling n_{slices} we can avoid this value from going over the memory we wish the

Model	Type	Original Complexity	Details
Office	Polygonal	17.353 pts.	Office environment with cubicles and hallways
Church	Polygonal	26.721 pts.	Simple church with stairs and columns
Sibenik	Polygonal	47.658 pts.	Cathedral of St. James on Sibenik, Croatia
Columns	Polygonal	143.591 pts.	Big environment with localized complexity.
Room	3D laser scan	271.731 pts.	3D Scan of a room with chairs and several objects.
Street	Polygonal	281.169 pts.	Outside street environment with an irregular floor, houses, and several objects
Entrance	3D laser scan	580.062 pts.	Entrance of the Batalha monastery in Portugal.

Table 1. Features of the models used for evaluation

application to consume, since all other memory required for the application is for tasks unrelated to collision detection.

We were interested in studying the direct behavior of our algorithm and did not wish to mask performance with asynchronous I/O threads. Results on collision detection have been verified through establishing a fixed route to navigate with the pawn avatar where it goes through different situations and scenarios, such as "climbing" single steps, "traversing" planar areas, going "near" cylindrical or square shaped columns and "falling" from a height. Whilst the number of created collision maps for a scene can affect collision results, the actual number of buffer slices n_{slices} will only affect potentially the performance, as the system checks all slices at the avatar´s current height. Tests on Cathedral 700x700 with 130 slices and n_{slices} set to 10 have showed us that reading from the disk at runtime has a bigger impact on efficiency than storing a higher number of images on commodity RAM. For instance, when using a maximum buffer of 10 slices and reading a high resolution image from the disk, we notice a sudden drop in frame-rate (Figure 6), and this can also be noticed when the pawn falls from a higher structure, and needs to rapidly load a series of maps on his way to the ground. By increasing n_{slices} to 16 on this scenario, we ensure the storage of enough slices to represent the ground floor and the platform on top of the steps (Experiment B in the accompanying video). Little difference was noticed on memory load (5,33MB), and the interaction was much smoother.

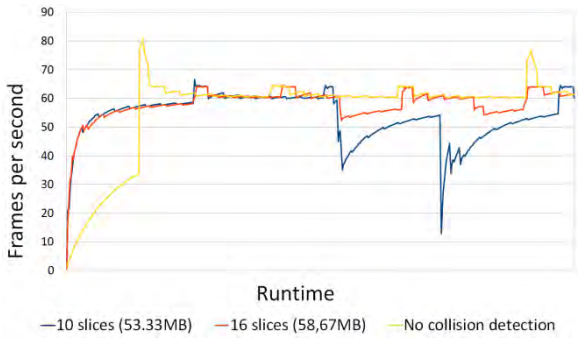


Figure 6. Average frame rate and memory consumption comparison between different n_{slices} configurations with 700x700 images, and no collision detection scenario. (Experiment B, Cathedral)

Results also show that our algorithm did not affect in any way the rendering speed of the interactive application. Figure 6 shows that the frame-rate was nearly identical in the same scenario with and without collision detection using 16 slices.

Although we did not aim for high precision on collision response, our technique has presented precise results on different resolutions. We note that point clouds are themselves approximations to surfaces, and as such a collision amongst points is an unlikely event, Figueiredo et. al use the average closest point to point distance divided by two to establish a conservative bounding box around each point, which can generate false collisions when close. With our approach, instead of a box, we use the pixels of a zoomed out view which is also conservative. Precision results with the different algorithms were verified empirically by changing the

Model	Time (s)	Total Complexity	Memory
Office	41,23	9.349.585 pts	610,3 MB
Church	58,14	6.475.125 pts	536,58 MB
Sibenik	78,87	5.199.093 pts	532,48 MB
Columns	42,92	2.612.303 pts	241,66 MB
Street	114,75	7.142.361 pts	598,02 MB
Entrance	13,9	580.062 pts.	122,88 MB
Room	6,96	271.731 pts.	67,58 MB

Table 2. Time, Total complexity (with generated points) and Memory consumption on the pre-processing stage for a 700x700

	Image-based (700x700)	BVH Oct 4096
Frame-rate	30 fps	16 to 30 fps
Total Memory	143.36 MB	225,44 MB
Pre. proc. time	13.9 s	1500 s

Table 3. Comparison between point-cloud techniques (Experiment A)

color of the avatar when a collision occurs. We found that using collision maps with a resolution of 700x700 enabled one to lower the number of false collisions from other methods when close to obstacles.

Floor collision has been performed perfectly in all resolutions, since its precision is defined by the rgb value of the point. Collisions with obstacles are more affected by resolution as is to be expected, since we rely on pixel finesse to precisely know the position of a given surface. Tests on office and street (Figure 7) have showed the same errors of small object interference or fake collisions due to diffuse information about object boundaries. These are more noticeable on the interactions with round objects on Street (Figure 7) where we can notice the aliasing creating invisible square barriers around a perfectly round object.

Table 3 compares our technique with the work from Figueiredo et. al [MJBJ], which has been tested on one of our experimental scenarios (Figure 7), the walkthrough in the point cloud of the Batalha Monastery (Experiment A in the accompanying video, 700x700 resolution n_{slices}set to 10), on a machine with a slightly faster processing speed and RAM than the one used for our walkthroughs. We compared our results with their best performance alternative, that bases the surface partition on 4096 cells of the octree.

Frame-rate was disturbed during the collision detection process on the R-tree approach, while it remained steady at the 30 fps during the whole execution of our application. Also, the image-based technique has required much less memory to be executed, even with a significantly high number of slices loaded in memory. The biggest difference is in the pre-processing times. Our approach was executed 107 times faster than the BVH approach. The pre-processing stage must only be performed once for each configuration, since the representation is written and loaded to the hard-drive for further interactions.

As stated in section 2 the research on point cloud collision detection is recent, and non-existent regarding image-based techniques. Our pioneer solution has presented excellent results, not only performing better than other works on point clouds published in the scientific community, but also being flexible enough to be applied on models from CAD, or combined with precise collision response techniques. Our technique can be performed with our representation on any computer that can render the input model at an acceptable frame-rate, without requiring much processing from the CPU or GPUs.

5. CONCLUSION AND FUTURE WORK

A new image-based environment representation for collision detection has been presented, using 2.5+D slices of an environment or buildings across the z axis. These images contain at each pixel, information about a given voxel, representing i's contents with colors. Height map information is stored on the red channel, and obstacles are indicated on the blue channel. This allows us to have a Broad phase collision detection stage that is performed with high efficiency and scalability, where the user can choose the precision according to the computing power at hand by simply adjusting the resolution of the produced images. Point clouds and polygonal models are ubiquitously processed, making our approach currently the best suited for fast interaction with massive laser-scan data. This work fills a gap in the area of collision detection, exploring a scenario that has been receiving more attention recently.

Future Work

Using graphic card capabilities such as the stencil buffer for broad-phase collision detection and vertex shaders for point coloring could greatly speed up these operations, and also, calculations would be moved to the GPU, taking away some work from the CPU. Applying this representation of environments also on objects of the scene, or even applying it to the avatars we're using on the interaction, could present interesting results. Using non-uniform resolution images on environments where we do not have a uniform complexity, would also help us achieve more precision on our narrow phase, or on these presented situations.

Image comparison techniques and compression can also be applied to the generated images in order to decrease the number of times we need to load a slice, and also the number of collision detection checks we must do. In several man-made structures such as buildings, many slices tend to be identical; intra-slice compression also presents itself as an interesting avenue of research.

6. ACKNOWLEDGMENTS

We would like to thank Artescan for the point clouds provided.

7. REFERENCES

[BHMTMG] Bruno Heidelberger, Matthias Teschner, and Markus Gross, Real-Time Volumetric Intersections of Deforming Objects, Proceedings of Vision, Modeling, and Visualization 2003,461-468,2003

[BMM] Brian Mirtich, V-clip: fast and robust polyhedral collision detection, ACM Trans. Graph., 17:177--208, July 1998.

[CLFTYC] Céline Loscos, Franco Tecchia, and Yiorgos Chrysanthou, Real-time shadows for animated crowds in virtual cities, In Proceedings of the ACM symposium on Virtual reality software and technology, VRST '01, pages 85--92, New York, NY, USA, 2001. ACM.

[DKDP] Dave Knott and Dinesh K. Pai, Cinder - collision and interference detection in real-time using graphics hardware, Proceedings of Graphics Interface, pages 73--80, 2003.

[FSJF] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou, Image-based collision detection and response between arbitrary volume objects, In Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08, pages 155--162, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[GBWHS] G. Baciu, Wingo Sai-Keung Wong, and Hanqiu Sun, Recode: an image-based collision detection algorithm, In Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference on, pages 125 --133, oct 1998.

[GBWSW] George Baciu and Wingo Sai-Keung Wong, Hardware-assisted self collision for deformable surfaces, Proceedings of the ACM symposium on Virtual reality software and technology, 2002.

[JFHFCP] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G. Kry, Volume contact constraints at arbitrary resolution, ACM Trans. Graph., 29:82:1--82:10, July 2010.

[JKGZ] Jan Klein and Gabriel Zachmann, Point cloud collision detection, Computer Graphics Forum, 23(3):567--576, 2004.

[JSLR] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski, Layered depth images, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998.

[KMOOTK] Karol Myszokowski, Oleg G. Okunev, and Tosiyasu L. kunii, Fast collision detection between complex solids using rasterizing graphics hardware, The Visual Computer, 11(9):497 -- 512, 1995.

[MJBJ] Mauro Figueiredo, João Oliveira, Bruno Araújo, and João Pereira, An efficient collision detection algorithm for point cloud models, 20th International conference on Computer Graphics and Vision, 2010.

[MLJC] M.C. Lin and J.F. Canny, A fast algorithm for incremental distance calculation, In Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, pages 1008 --1014 vol.2, apr 1991.

[MT] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, and W. Strasser, Collision detection for deformable objects, Computer Graphics Forum,24(1):61--81,2005

[NBJM] Niels Boldt and Jonas Meyer, Self-intersections with cullide, Eurographics, 23(3), 2005.

[NMAS] Noralizatul Azma Bt Mustapha Abdullah, Abdullah Bin Bade, and Sarudin Kari, A review of collision avoidance technique for crowd simulation, 2009 International Conference on Information and Multimedia Technology, (2004):388--392, 2009.

[NSRMLDM] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha, Cullide: interactive collision detection between complex models in large environments using graphics hardware, In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '03, pages 25--32, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[PMH] Philip M. Hubbard, Approximating polyhedra with spheres for time-critical collision detection, ACM Transactions on Graphics, 15(3):179--210, July 1996.

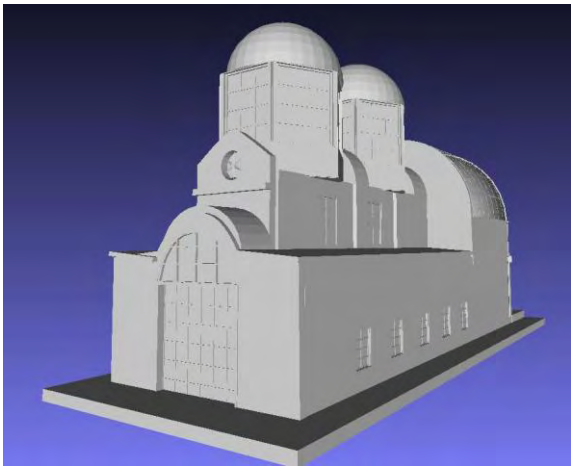
[SGMCLDM] S. Gottschalk, M. C. Lin, and D. Manocha. 1996. OBBTree: a hierarchical structure for rapid interference detection. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH '96). ACM, New York, NY, USA, 171-180.

[SKMNFF] Stephan Kimmerle, Matthieu Nesme, and François Faure, Hierarchy Accelerated Stochastic Collision Detection, In 9th International Workshop on Vision, Modeling, and Visualization, VMV 2004, pages 307-312, Stanford, California, États-Unis, November 2004.

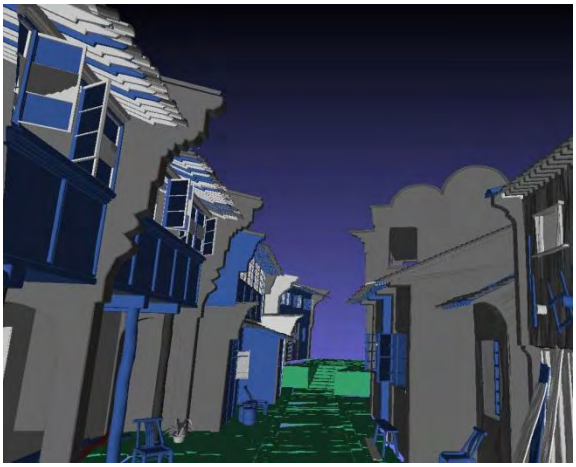
[SKTGKICB] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and Richard Rowe, Collision detection: A survey, In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pages 4046 --4051, oct. 2007.

[TLTAM] Thomas Larsson and Tomas Akenine-Möller, Collision detection for continuously deforming bodies, Eurographics 2001.

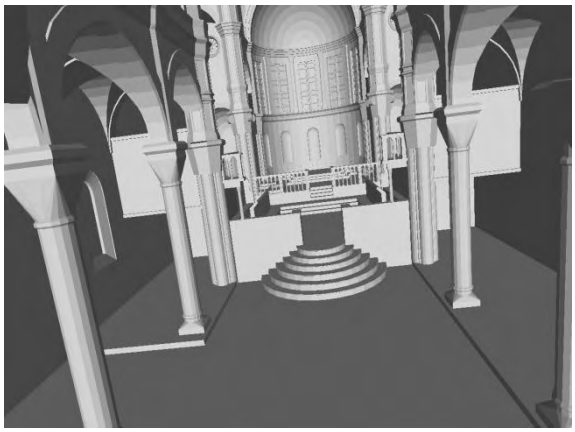
[XZYJK] Xinyu Zhang and Y.J. Kim, Interactive collision detection for deformable models using streaming aabbs, Visualization and Computer Graphics, IEEE Transactions on, 13(2):318 --329, march-april 2007.



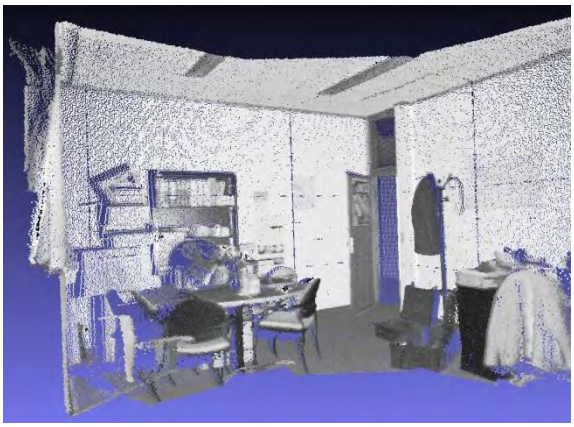
(a) Church



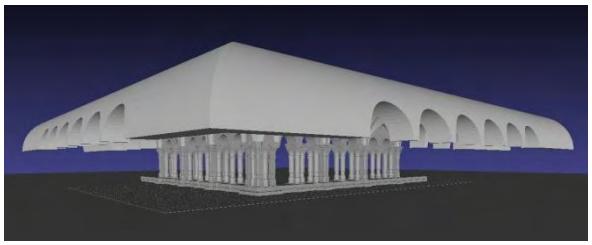
(b) Street



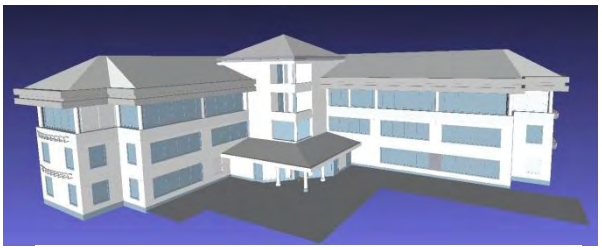
(c) Sibenik



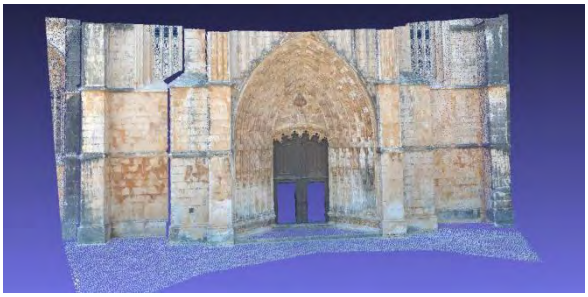
(d) Room



(e) Columns



(f) Office



(e) Batalha

Figure 7. Pictures of all the tested input polygonal models and point clouds. Environments with different topologies were chosen for this purpose.

Color Preserving HDR Fusion for Dynamic Scenes

Gökdeniz Karadağ
Middle East Technical University, Turkey
gokdeniz@ceng.metu.edu.tr

Ahmet Oğuz Akyüz
Middle East Technical University, Turkey
akyuz@ceng.metu.edu.tr

ABSTRACT

We present a novel algorithm to efficiently generate high quality high dynamic range (HDR) images. Our method is based on the idea of expanding the dynamic range of a reference image at granularity of tiles. In each tile, we use data from a single exposure, but different tiles can come from different exposures. We show that this approach is not only efficient and robust against camera and object movement, but also improves the color quality of the resulting HDR images. We compare our method against the commonly used HDR generation algorithms.

Keywords: High dynamic range imaging, image fusion, color quality

1 INTRODUCTION

The interest in HDR imaging has rapidly gained popularity in recent years. This has been accompanied by the development of various methods to create HDR images. While it is believed that using dedicated HDR capture hardware will be the de-facto way of generating HDR images in future [Rei10a], software solutions are still commonly used in today's systems. Among these multiple exposure techniques (MET) are the most dominant [Man95a, Deb97a].

In METs, several images of the same scene are captured by varying the exposure time between the images. This ensures that each part of the captured scene is properly exposed in at least one image. The individual images are then merged to obtain the HDR result. Although variations exist, the equation below is typically used for the merging process:

$$I_j = \sum_{i=1}^{N} \frac{f^{-1}(p_{ij})w(p_{ij})}{t_i} \Big/ \sum_{i=1}^{N} w(p_{ij}). \qquad (1)$$

Here N is the number of LDR images, p_{ij} is the value of pixel j in image i, f is the camera response function, t_i is the exposure time of image i, and w is a weighting function used to attenuate the contribution of poorly exposed pixels.

In Equation 1, a weighted average is computed for every pixel. While this may be desirable for attenuating noise, it introduces unwanted artifacts due to ghosting and misalignment problems. In this paper, we show that this approach also results in the desaturation of colors making the HDR image less saturated than the its constituent exposures.

Computing a weighted average for every pixel also requires that the individual pixels are perfectly aligned. Otherwise, pixels belonging to different regions in the scene will be accumulated resulting ghosting and alignment artifacts.

In this paper, we propose a method that largely avoids both of these problems. Our method is underpinned by the idea that instead of computing an average for every pixel, one can use the pixels from a single properly exposed image. A different image can be used for different regions ensuring that the full dynamic range is captured. We also introduce the concept of working in tiles instead of pixels to make the algorithm more robust against local object movements.

2 PREVIOUS WORK

Starting with the pioneering works of Madden [Mad93a] and Mann and Picard [Man95a], various algorithms have been developed to create HDR images. The early work focused on recovering the camera response function and choosing an appropriate weighting function [Deb97a, Mit99a, Rob03a, Gro04a]. These algorithms assumed that the exposures that are used to create an HDR image are perfectly aligned and the scene is static.

Ward developed a method based on median threshold bitmaps (MTBs) to allow photographers use hand-held images of static scenes in HDR image generation [War03a]. His alignment algorithm proved to be very successful and is used as an initial step of more advanced alignment and ghost removal algorithms [Gro06a, Jac08a, Lu09a].

In another alignment algorithm, Cerman and Hlaváč estimated the initial shift amounts by computing the correlation of the images in the Fourier domain [Cer06a]. This, together with the initial rotational estimate which

was assumed to be zero, was used as a starting point for the subsequent iterative search process.

Tomaszewska and Mantiuk employed a modified scale invariant feature transform (SIFT) [Low04a] to extract local features in the images to be aligned [Tom07a]. The prominent features are then selected by the RANSAC algorithm [Fis81a]. This refined set of features are then used to compute a homography between the input images.

Several methods have been proposed to deal with ghosting artifacts. These algorithms usually pre-align the input exposures using MTB or other algorithms to simplify the ghost detection process. Some of these algorithms avoid merging suspicious regions where there is high variance [Kha06a, Gal09a, Ram11a]. Other algorithms try to detect the movement of pixels and perform pixel-wise alignment [Zim11a]. A recent review of HDR ghost removal algorithms can be found in Srikantha and Sidibé [Sri12a].

There are also existing algorithms that attempt to combine data from multiple exposures for the purpose of generating a single low dynamic range (LDR) image. Among these, Goshtasby first partitions the images into tiles [Gos05a]. For each tile, he then selects the image that has the highest entropy. The tiles are blended using smooth blending functions to prevent seams. Mertens et al., on the other hand, do not use tiles but utilize three metrics namely contrast, saturation, and well-exposedness to choose the best image for each pixel [Mer07a]. Similar to Goshtasby, Várkonyi-Kóczy et al. propose a tile based algorithm where tiles are selected to maximize detail using image gradients [Var08a]. In another tile based algorithm, Vavilin and Jo use three metrics; mean intensity, intensity deviation, and entropy to choose the best exposure for each tile [Vav08a]. In contrast to previous tile based studies, they choose tile size adaptively based on local contrast. Finally, Jo and Vavilin propose a segmentation based algorithm which allows choosing different exposures for different clusters [Jo11a]. Unlike previous methods they use bilateral filtering during the blending stage.

It is important to note that existing tile-based algorithms attempt to generate LDR images with more details and enhanced texture information, whereas our goal is to generate HDR images with natural colors. Our approach alleviates the need for explicit ghost detection and removal procedures. If the dynamic parts of a scene do not span across regions with significantly different luminance levels, no ghost effects will occur in the output. Also, we avoid redundant blending of pixels that can result in reduced color saturation.

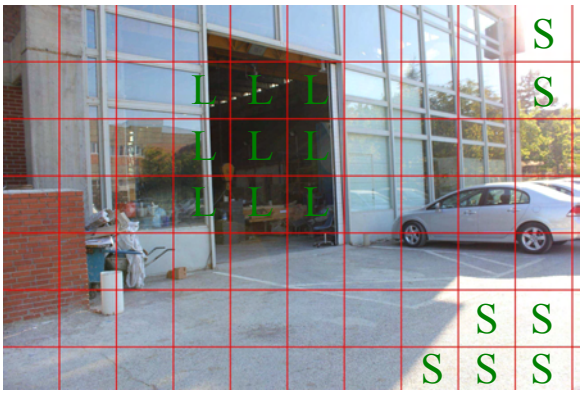


Figure 1: We partition the images into tiles and determine which exposure to use for each tile.

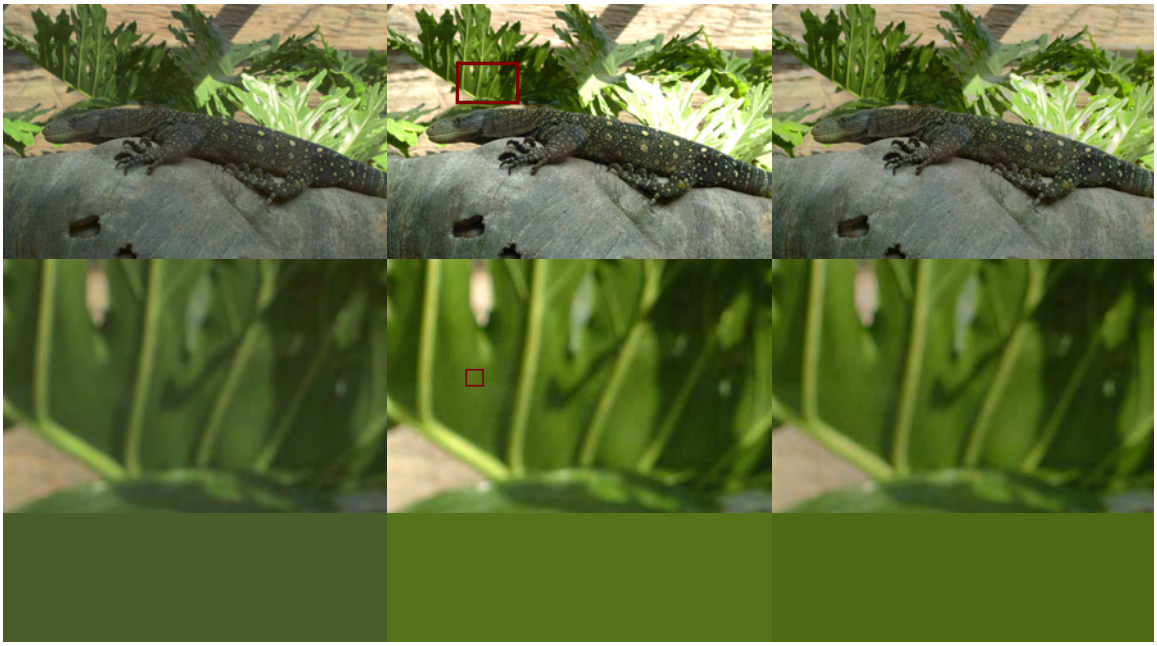
3 ALGORITHM

The first step of our algorithm is to align the input exposures using the MTB algorithm [War03a]. In this part, both the original MTB or the MTB with the rotation support can be used.

Once the images are aligned, we partition each exposure into tiles. Our goal then becomes to choose the best image that represents the area covered by each tile. A sample image is shown in Figure 1 to illustrate this idea. In this image, the under-exposed tiles are marked with **L** indicating that these tiles should come from a longer exposure. Similarly, over-exposed regions are marked by **S** suggesting that shorter exposures should be used for these tiles. Unmarked tiles can come from the middle exposures.

To make these decisions, we need to define a quality metric that indicates whether a tile is well-exposed. To this end, we experimented with the mean intensity as well as the number of under- and over-exposed pixels within a tile as potential metrics. Our results suggested that using the mean intensity gives better results. Therefore, we marked a tile as a *good* tile if its mean intensity is in the range $[I_{min}, I_{max}]$. I_{min} and I_{max} are user parameters, but we found that $I_{min} = 50$ and $I_{max} = 200$ can be used as reasonable defaults.

Based on this criteria, we compute the number of good tiles for each exposure. We choose the exposure with the maximum number of good tiles as the reference exposure. This exposure serves as the *donor* which provides data for all tiles whose mean intensity stays in the aforementioned limits. This leniency allows us to use the same image as much as possible and provides greater spatial coherency. For the remaining tiles, we choose the second reference exposure and fill in the tiles which are valid in this exposure. This process is



(a) Standard MET (b) LDR reference (c) Our result

Figure 2: (a) HDR image created by using the standard MET. (b) Selected individual exposure from the bracketed sequence. (c) HDR image created using our algorithm. The top row shows the full images. The middle row shows the close-up view of a selected region. The bottom row shows the color of a single pixel from the region indicated in the middle row. Both HDR images are tone mapped using the photographic tone mapping operator [Rei02a]. As can be seen in the zoomed views, the color quality of our result is closer to the selected reference image.

recursively executed until a source image is found for all tiles[1]. This process can be represented as:

$$I_j = \sum_{i=1}^{N} \frac{f^{-1}(p_{ij})W_{ij}}{t_i}, \tag{2}$$

$$W_{ij} = \begin{cases} 1 & \text{if pixel } j \text{ comes from image } i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that we no longer have the $w(p_{ij})$ term from Equation 1 as we do not compute a weighted average.

Finally, we use a blending strategy to prevent the visibility of seams at tile boundaries. For this purpose, we create Gaussian pyramids of weights (W_{ij}) and Laplacian pyramids of source images. We then merge the images by using Equation 2 at each level of the pyramid and collapse the pyramid to obtain the final HDR image. We refer the reader to Burt and Adelson's original paper for the details of this process [Bur83a].

Since the tiles are not overlapping our algorithm ensures that within each tile data from only a single source image is used. As we demonstrate in the next section, this improves the color saturation of the resulting HDR images. A second observation is that each tile is spatially coherent. This means that motion related artifacts will not occur within tiles. However, such artifacts can still occur across tiles. Thus our algorithm reduces the effect of motion artifacts but does not completely eliminate them.

[1] It is possible that the a tile is under- or over-exposed in all input images. In this case, we choose the longest exposure if the tile is under-exposed and shortest exposure otherwise.

4 RESULTS AND ANALYSIS

We present the results of our color preserving HDR fusion algorithm under three categories namely: (1) Fixed camera & static scene, (2) hand-held camera & static scene, and (3) hand-held camera & dynamic scene. For the first configuration, we illustrate that the color quality of the HDR image created by our method is superior to the output of the standard HDR fusion algorithm shown in Equation 1. A sample result for this case is depicted in Figure 2 where the output of the standard MET is shown on the left and our result is shown on the right. A selected exposure from the bracketed sequence is shown in the middle for reference.

For the image on the left, we used the tent weighting function proposed by Debevec and Malik [Deb97a]. We used the sRGB camera response function for both images, and a tile size of 64×64 for our result. It can be seen that, due to the pixel-wise averaging process, the output of the standard MET has a washed-out appearance. Our result, on the other hand, is colorimetrically closer to the selected exposure. This is a consequence of avoiding unnecessary blending between images.

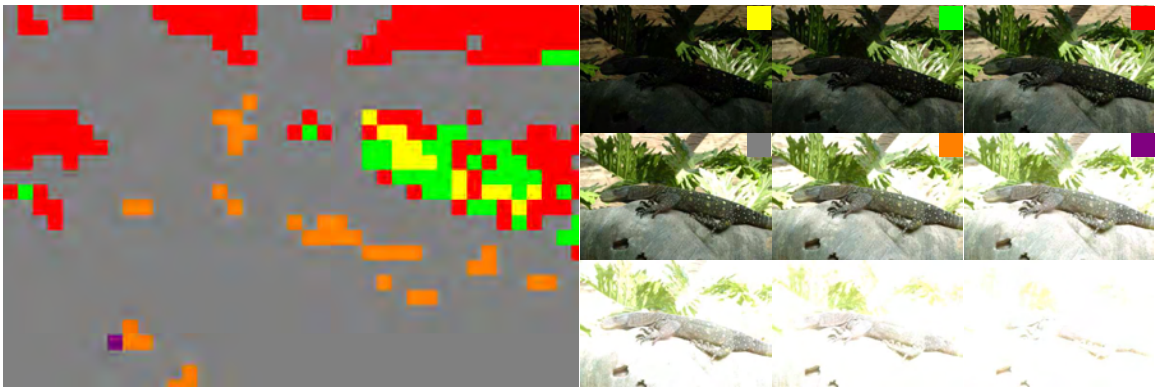


Figure 3: The colors show the correspondence between the tiles in the HDR image and the source images that they were selected from. We can see that most tiles were selected from the fourth image. Figure courtesy of Erik Reinhard [Rei10a].

Figure 3 shows which tiles in the output HDR image came from which images in the exposure sequence. The correspondence is shown by color coding the individual exposures. As we can see from this figure, the majority of the tiles were selected from the fourth exposure. The tiles that correspond to the highlights on the plants came from the darker exposures. On the other hand, the tiles that correspond to the crevices on the rock and the shadow of the lizard came from the lighter exposures. We can also see that the last three exposures were not used at all.

At this point, it would be worthwhile to discuss why the standard MET gives rise to a washed-out appearance and our algorithm does not. We would not expect to see this problem if all exposures were perfect representations of the actual scene. However, in reality, there are slight differences between exposures that are not only due to changing the exposure time. Slight camera movements, noise, and inaccuracies in the camera response curve can all cause variations between the actual observations. The combined effect of these variations result in reduced color saturation. By avoiding unnecessary blending, we also avoid this artifact.

The second test group consists of images of a static scene captured by a hand-held camera (Figure 4). In this figure, the left column shows the unaligned result created by directly merging five bracketed exposures. The middle column shows the tone mapped HDR output after the exposures are aligned by using the MTB algorithm. The right column shows our result obtained by first aligning the exposures using the MTB algorithm, and then merging them using our tile-based technique. As can be seen from the fence and the sign in the insets, our result is significantly sharper than that of the MTB algorithm. However, we also note that small artifacts are visible in our result on the letters “R” and “E”. Further examination reveals that these artifacts are due to using tiles from different exposures that are not perfectly aligned.

As the color map indicates, the majority of the final HDR image is retrieved from the exposure coded by red (exposures not shown). The darker regions retrieved data from the lighter (gray) exposure. The highlights at the top left corner received data from the darker (green) exposure. In fact, in this example, all five exposures contributed to the final image but the majority of the contribution came from these three exposures.

In the final category, we demonstrate the performance of our algorithm using scenes that have both global and local movement. To this end, we used the hdrgen software[2] which implements the MTB alignment algorithm and a variance based ghost removal method explained in Reinhard et al. [Rei10a]. In Figure 5, the left column shows the output obtained by only image alignment but without ghost removal. The middle column shows the result of alignment and ghost removal. Although the majority of the ghosts are removed, some artifacts are still visible on the flag as shown in the close-ups. The right column shows our result where these artifacts are eliminated. The color map indicates the source images for different regions of the HDR image.

We also demonstrate a case where our algorithm introduces some unwanted artifacts in high contrast and high frequency image regions as the window example in Figure 6. The bright back light and window grates cause high contrast. If the tile size is large, blending tiles from different exposures produces sub-par results. A reduced tile size eliminates these artifacts.

Our choice of prioritizing the reference image increases success in image sets where ghosting effects would normally occur. If the object movements are located in regions with similar lighting conditions, our algorithm prefers the image closer to reference image while constructing tiles, preventing ghosting effects. It is possible that an object moves between regions of different lighting conditions, and our algorithm may choose tiles

[2] `http://www.anyhere.com`

Figure 4: Left: Unaligned HDR image created from hand-held exposures. Middle: Exposures aligned using the MTB algorithm. Right: Our result. The close-ups demonstrate that our algorithm produces sharper images. The color map shows the source exposures for different regions of the HDR image.

(a) Alignment (b) Alignment and ghost removal (c) Our result

Figure 5: Left: Aligned HDR image created from hand-held exposures using the MTB algorithm. Middle: Aligned and ghost removed HDR image. Right: Our result. The insets demonstrate that ghosting artifacts are eliminated in our result. The color map shows the source exposures for different regions of the HDR image.

from different images where the moving object can be seen. In this case different copies of the object may be present in multiple locations in the output image.

Finally, we report the running times of our algorithm. An unoptimized C++ implementation of our algorithm was able to create high resolution (18 MPs) HDR images from 9 exposures within 30 seconds including all disk read and write times. We conducted all of our test on an Intel Core i7 CPU running at 3.20 GHz and equipped with 6 GBs of memory. This suggests that our algorithm is practical and can easily be integrated into existing HDRI workflows.

5 CONCLUSIONS

We presented a simple and efficient algorithm that improves the quality of HDR images created by using multiple exposures techniques. By not redundantly averaging pixels in low dynamic regions, our algorithm preserves the color saturation of the original exposures, and reduces the effect of ghosting and alignment artifacts. As future work, we are planning to make the tiling process adaptive instead of using a uniform grid. This would prevent artifacts that can be caused by sudden illumination changes between neighboring tiles coming from different exposures. We are also planning to perform blending using edge-aware Laplacian pyramid [Par11a] to avoid blending across sharp edges. Improved quality of our results can also be validated by a user study.

ACKNOWLEDGMENTS

This work was partially supported by METU BAP-08-11-2011-123.

6 REFERENCES

[Bur83a] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4):532 – 540, apr 1983.

Figure 6: Top: A tone-mapped HDR image with 128x128 tile size. Tile boundaries are highly visible in the close-up. Middle: Changing the tile size to 32x32 removes most of the artifacts, but some remain in diagonal lines. Bottom: Using a 2x2 tile size eliminates remaining artifacts.

[Cer06a] Lukáš Cerman and Václav Hlaváč. Exposure time estimation for high dynamic range imaging with hand held camera. In *Computer Vision Winter Workshop, Czech Republic*, 2006.

[Deb97a] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97 Conf. Proc.*, pages 369–378, August 1997.

[Fis81a] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[Gal09a] O. Gallo, N. Gelfandz, Wei-Chao Chen, M. Tico, and K. Pulli. Artifact-free high dynamic range imaging. In *Computational Photography (ICCP), 2009 IEEE International Conference on*, pages 1 –7, april 2009.

[Gos05a] A. Ardeshir Goshtasby. Fusion of multi-exposure images. *Image and Vision Computing*, 23(6):611 – 618, 2005.

[Gro04a] M.D. Grossberg and S.K. Nayar. Modeling the space of camera response functions. *Pattern Analysis and Machine Intelligence, IEEE Trans. on*, 26(10):1272 –1282, 2004.

[Gro06a] Thorsten Grosch. Fast and robust high dynamic range image generation with camera and object movement. In *Proc. of Vision Modeling and Visualization*, pages 277–284, 2006.

[Jac08a] Katrien Jacobs, Celine Loscos, and Greg Ward. Automatic high-dynamic range image generation for dynamic scenes. *IEEE CG&A*, 28(2):84–93, 2008.

[Jo11a] K.H. Jo and A. Vavilin. Hdr image generation based on intensity clustering and local feature analysis. *Computers in Human Behavior*, 27(5):1507–1511, 2011.

[Kha06a] Erum Arif Khan, Ahmet Oğuz Akyüz, and Erik Reinhard. Ghost removal in high dynamic range images. *IEEE International Conference on Image Processing*, 2006.

[Low04a] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[Lu09a] Pei-Ying Lu, Tz-Huan Huang, Meng-Sung Wu, Yi-Ting Cheng, and Yung-Yu Chuang. High dynamic range image reconstruction from hand-held cameras. In *CVPR*, pages 509–516, 2009.

[Mad93a] B. C. Madden. Extended dynamic range imaging. Technical report, GRASP Laboratory, Uni. of Pennsylvania, 1993.

[Man95a] S Mann and R Picard. Being 'undigital' with digital cameras: Extending dynamic range by combining differently exposed pictures, 1995.

[Mer07a] T. Mertens, J. Kautz, and F. Van Reeth. Exposure fusion. In *Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on*, pages 382 –390, 29 2007-nov. 2 2007.

[Mit99a] T. Mitsunaga and S. K. Nayar. Radiometric self calibration. In *Proceedings of CVPR*, volume 2, pages 374–380, June 1999.

[Par11a] Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. Local laplacian filters: edge-aware image processing with a laplacian pyramid. *ACM Trans. Graph.*, 30(4):68:1–68:12, July 2011.

[Ram11a] Shanmuganathan Raman and Subhasis Chaudhuri. Reconstruction of high contrast images for dynamic scenes. *The Visual Computer*, 27(12):1099–1114, 2011.

[Rei02a] Erik Reinhard, Michael Stark, Peter Shirley, and Jim Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics*, 21(3):267–276, 2002.

[Rei10a] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*. Morgan Kaufmann, San Francisco, second edition edition, 2010.

[Rob03a] Mark A. Robertson, Sean Borman, and Robert L. Stevenson. Estimation-theoretic approach to dynamic range enhancement using multiple exposures. *Journal of Electronic Imaging 12(2), 219 228 (April 2003).*, 12(2):219–228, 2003.

[Sri12a] Abhilash Srikantha and Désiré Sidibé. Ghost detection and removal for high dynamic range images: Recent advances. *Signal Processing: Image Communication*, (0):–, 2012.

[Tom07a] Anna Tomaszewska and Radoslaw Mantiuk. Image registration for multi-exposure high dynamic range image acquisition. In *WSCG: Proc. of the 15th Intl. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2007.

[Var08a] A. R. Varkonyi Koczy, A. Rovid, and T. Hashimoto. Gradient-based synthesized multiple exposure time color hdr image. *Instrumentation and Measurement, IEEE Transactions on*, 57(8):1779 –1785, aug. 2008.

[Vav08a] A. Vavilin and K.H. Jo. Recursive hdr image generation from differently exposed images based on local image properties. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 2791–2796. IEEE, 2008.

[War03a] Greg Ward. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *Journal of Graphics Tools*, 8(2):17–30, 2003.

[Zim11a] Henning Zimmer, Andrés Bruhn, and Joachim Weickert. Freehand hdr imaging of moving scenes with simultaneous resolution enhancement. *Computer Graphics Forum*, 30(2):405–414, 2011.

On-the-fly Luminance Correction for Rendering of Inconsistently Lit Point Clouds

Thomas Kanzok
Chemnitz University of Technology
Department of Computer Science
Visual Computing Laboratory
Straße der Nationen 62
09111 Chemnitz, Germany
thomas.kanzok@informatik.tu-chemnitz.de

Lars Linsen
Jacobs University
School of Engineering & Science
Visualization and Computer Graphics Laboratory
Campus Ring 1
28759 Bremen, Germany
l.linsen@jacobs-university.de

Paul Rosenthal
Chemnitz University of Technology
Department of Computer Science
Visual Computing Laboratory
Straße der Nationen 62
09111 Chemnitz, Germany
paul.rosenthal@informatik.tu-chemnitz.de

ABSTRACT

Scanning 3D objects has become a valuable asset to many applications. For larger objects such as buildings or bridges, a scanner is positioned at several locations and the scans are merged to one representation. Nowadays, such scanners provide, beside geometry, also color information. The different lighting conditions present when taking the various scans lead to severe luminance artifacts, where scans come together. We present an approach to remove such luminance inconsistencies during rendering. Our approach is based on image-space operations for both luminance correction and point-cloud rendering. It produces smooth-looking surface renderings at interactive rates without any preprocessing steps. The quality of our results is similar to the results obtained with an object-space luminance correction. In contrast to such an object-space technique the presented image-space approach allows for instantaneous rendering of scans, e.g. for immediate on-site checks of scanning quality.

Keywords

Point-cloud Rendering, Image-space Methods, Luminance Correction, Color-space Registration

1. INTRODUCTION

In the field of civil engineering large structures like bridges have to be surveyed on a regular basis to document the present state and to deduct safety recommendations for repairs or closures. Typically this is done by measuring the structures manually or semiautomatically at predefined measuring points and adding detailed photographs or by using completely automatic 3D scanning techniques.

Nowadays, both dominant surface digitalization techniques, laser scanning [BR02] as well as photogrammetry [SSS06,TS08], produce colored point clouds where the color of each point matches the color of the respective surface part under the lighting conditions at the time of scanning. Many photographs become redundant with this additional information. However, when dealing with large structures one has to do several scans from different points of view in order to generate a complete building model of desired resolution. As in most cases only one 3D scanner is used and relocated for each scan, the time of day and therefore the lighting conditions may differ significantly between adjacent scans or, on a cloudy day, even within one scan.

When combining the different scans to one object representation and using a standard geometry-based registration approach, the resulting point cloud may have severe inconsistencies in the luminance values. Because of the scanning inaccuracies in the geometric measures, the renderings of registered scans exhibit disturbing patterns of almost randomly changing luminance assignment in regions where scans with different lighting conditions overlap. We present an approach to correct the luminance and create a consistent rendering. "Consistent" in this context

means that no local luminance artifacts occur; it does not mean that the global illumination in the rendering is consistent.

A simple, yet effective approach to adjust the luminance is to average the luminances locally within neighborhoods in object space, as outlined in Section 4. We use this approach as a reference to the image-space approach we propose in Section 5. The reason for introducing the image-space approach is that the luminance correction in object space is rather time-consuming and needs to be done in a preprocessing step. The engineers, however, would like to get an immediate feedback during their field trip whether the scans they have taken are capturing all important details and are of sufficient quality. Hence, an immediate rendering of the combined scans is required. Our image-space approach allows for such an instantaneous investigation of the scans since it is capable of producing high-quality renderings of inconsistently lit point clouds at interactive framerates.

2. RELATED WORK

Today both the amount and size of generated surface data are steadily increasing. Beginning with the Digital Michelangelo project [LPC+00], which was the first one generating massive point clouds, the scanning hardware was getting significantly cheaper while producing results of increasing resolution and quality. The datasets that are typically generated these days range from hundreds of million to several billion surface points [WBB+08].

In this setting it is obvious, that global reconstruction of the surface becomes infeasible and the use of local reconstruction techniques, like splatting [LMR07, PzvBG00, RL00] or implicit reconstruction [AA03, ABCO+03, GG07], has become state of the art. However, these approaches still need some preprocessing steps, constricting instant preview of generated data. With the advent of image-space point-cloud rendering techniques [DRL10, MKC07, RL08, SMK07] it became possible to interactively render and explore scanned datasets on the fly without any preprocessing.

These new possibilities open up a whole set of new applications, but also induce new challenges. The sampling of the generated point clouds can be highly varying, making the use of several rendering approaches difficult. This can be circumvented by using level-of-detail methods conveying a nearly uniform sampling in image space [BWK02, GZPG10, RD10].

Registration of color scans, produced under different light conditions, can result in luminance inconsistencies of the resulting colored point cloud. Consequently, renderings of such point clouds exhibit significant high-frequency noise. Removing such noise has always been an important topic in image processing. There exists a vast amount of approaches in this field [BC04, BJ10, KS10, WWPS10], which typically try to remove noise in an image by analyzing the spatial neighborhood of a pixel and adjusting the pixel value accordingly. Adams et al. [AGDL09] propose a *k*d-tree-based filtering which is also able to handle geometry if connectivity information is given. This is not the case for point clouds resulting from standard scanning techniques. The aforementioned approaches are specialized on denoising images and do not utilize the particular nature of point-cloud renderings. A notable example of denoising that was explicitly designed for point clouds was presented by Kawata and Kanai [KK05], but it suffers from the restriction to only two different points for denoising.

We will show how to effectively exploit the whole amount of surface points that project to a pixel for interactively generating smooth renderings of inconsistently lit point clouds.

3. GENERAL APPROACH

Let P be a colored point cloud, i.e. a finite set of points $\mathbf{p} \in \mathbb{R}^3$ where each point is enhanced with RGB color information. Furthermore, we assume that colors stored at the surface points approximate the respective surface color except for luminance correctly. Our goal is to produce an interactive rendering of the point cloud with smoothly varying luminance, following the assumption that neighboring points represent surface parts with similar luminance.

To adjust the luminance of the point cloud we decided to use the HSV color model, since it naturally describes the luminance of a point $\mathbf{p}$ in its V component. Thus, we are able to manipulate luminance without interfering with hue and saturation. The basic approach is to convert the colors of all points to the HSV model, average the V component between selected surface points and convert the colors back to the RGB format for final rendering.

As a first idea one could think of prefiltering the whole point cloud in object space to achieve this goal. We implement this idea by generating a space partition for the point cloud, enabling the efficient generation of neighborhood information. Luminance of neighboring surface points is smoothed to generate a point cloud with smoothly varying luminance, see Section 4. This approach can effectively eliminate the luminance noise in point clouds when choosing a sufficiently large neighborhood. However, it takes a significant amount of precomputation time, especially for massive point clouds with hundreds of million

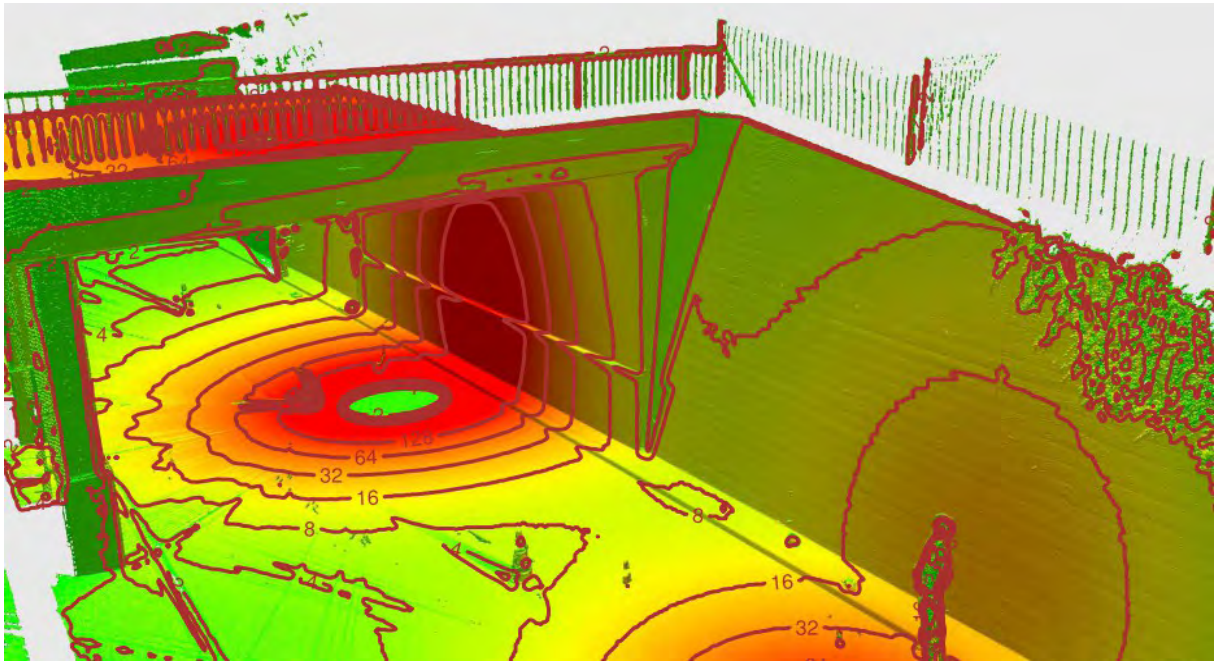


Figure 1. Image-space rendering of a real world point cloud. The number of projected points per pixel is color coded between 1 (green) and 150 (red), which is also emphasized by the isolines. The image was produced with applied depth thresholding and shading to enhance geometry perception.

points, which inhibits the instant rendering of generated point clouds.

To avoid this preprocessing step, we propose a GPU-assisted image-space luminance correction working on the fly. The approach utilizes the fact, that in most cases multiple surface points get projected to one pixel during rendering, as illustrated in Figure 1. When restricting the surface points to those representing non-occluded surface parts, a good approximation for the desired luminance can be obtained by averaging the luminance of the respective surface points. This is done in two rendering steps. In a first pass the scene is rendered to the depth buffer generating a depth mask. Following the idea of a soft z-buffer [PCD+97], an additional threshold ϵ is added to the depth mask, which defines the minimal distance between different consecutive surfaces. In a second render pass the depth mask is utilized to accumulate the luminance of all surface points, effectively contributing to a pixel. A detailed description of the method is given in Section 5.

After this step we apply image-space filters to fill pixels incorrectly displaying background color or occluded surface parts, as proposed by Rosenthal and Linsen [RL08]. The final rendering with associated depth buffer can be used to approximate surface normals per fragment, which opens up several possibilities for calculating postprocessing effects.

4. OFFLINE LUMINANCE CORRECTION

Luminance correction in object space requires the definition of a certain neighborhood for each surface point. We use the n nearest neighbors for each point as neighborhood. For fast detection of these neighborhoods, a three-dimensional kd-tree is built for the point cloud. Since luminance correction is done utilizing the HSV color space, all point colors are converted to this space. Then for each point we compute its neighbors and average their luminance. Finally the complete point cloud is converted back to RGB for rendering.

Note, that for weighted averaging also a kernel function can be used. However, several tests, e.g. with a Gaussian kernel, revealed no significant differences. Regarding the neighborhood size a value of $n=40$ has proven to produce appealing results. However, the precomputation time increases significantly with the number of points and number of neighbors. The luminance correction of a point cloud with 150 million surface points takes for example nearly six hours in an out-of-core implementation. Also when using an in-core implementation, the computation times are far too long for allowing instant views of generated point clouds.

5. IMAGE-SPACE LUMINANCE CORRECTION

Following the main idea of image-space point-cloud rendering, we propose an algorithm that facilitates high-quality point-cloud inspection without preprocessing, utilizing luminance correction in image space. The algorithm takes advantage of the fact that many points are projected to one pixel in models with high point densities, as already shown in Figure 1. Usually a large fraction of these points describe nearly the same geometric position on the surface. The other points belong to parts of the scene which are occluded by the surface closest to the viewer.

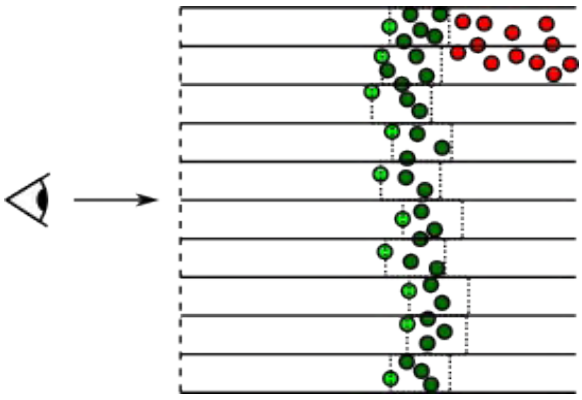
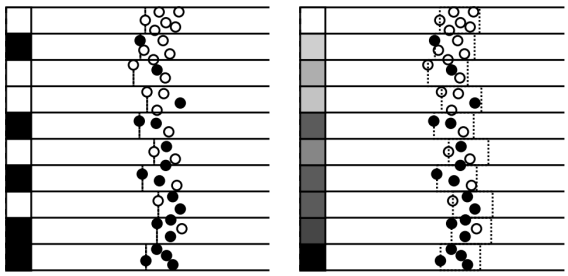


Figure 2. 2D illustration of point projection. The traditional z-buffer test renders only the nearest points (light green). Points representing occluded surface parts (red) but also redundant points on the same surface (dark green) are discarded. By increasing a fragment's z-value by a threshold ϵ (dotted lines) we still discard points from occluded surface parts but are able to blend the luminance of the remaining points for each pixel. (The view plane is the dashed line on the left-hand side.)

Our algorithm for correcting luminance in image space consists of two main rendering steps. In a first rendering pass a (linearized) depth map, selecting all points which represent visible surface parts, is generated (see Figure 2). In a second pass luminance is corrected for each pixel, taking all surface points into account which pass the depth-mask test.

For the first rendering pass all points are rendered resulting in a preliminary depth map and a texture T with the preliminary rendering result. The depth map is extended fragment-wise by the z-threshold, generating the desired, slightly displaced depth mask. Afterwards, we prohibit writing operations to the depth buffer such that every surface point that is closer than the value stored in the depth mask is drawn in the next render pass while farther points are discarded.



a) **b)**

Figure 3. Comparison of the rendering results with (a) normal z-buffering and (b) depth masked luminance correction. The change of predominant luminance at the surface points from top to bottom is rendered much smoother with luminance correction.

In the second render pass we accumulate the luminance of the rendered points using the OpenGL blending with a strictly additive blending function. All surface points are rendered to a non-clamped RGB floating point texture using the depth test. In the fragment shader we set the color of each fragment to (luminance,0,1), which produces a texture $\tilde{T}$ with the accumulated luminances in the R component and the number of blended points in the B component. Finally, we combine the blended texture $\tilde{T}$ with the preliminary rendering result T by converting T to the HSV color space, applying

$$T_{HSV} := (T_H, T_S, \frac{\tilde{T}_R}{\tilde{T}_B})$$

and converting the result back to the RGB color space. The result per pixel is a color with averaged luminance over all surface points, which passed the depth test, i.e. which belong to the visible surface.

Note that one can also easily average colors in RGB space by using a four-channel texture for $\tilde{T}$ and blending (R,G,B,1) for each fragment. Then the resulting color would be given by

$$T_{RGB} := (\frac{\tilde{T}_R}{\tilde{T}_A}, \frac{\tilde{T}_G}{\tilde{T}_A}, \frac{\tilde{T}_B}{\tilde{T}_A})$$

The difference between traditional z-buffering and our approach is depicted in Figure 3. Although our algorithm requires two rendering passes and therefore basically halves the framerate, we are able to produce smooth lighting much faster than with the preprocessing algorithm, making on-site preview feasible.

An enhancement to the base algorithm is to correct luminance not only in one pixel but to additionally use points from neighboring pixels. We do this by summing the luminance of a fragment in the final step over its 8-neighborhood and dividing by the total number of points. A Gaussian kernel can be used as additional weight for the points of the neighboring fragments to take their distances into account. This produces even smoother results than the simple

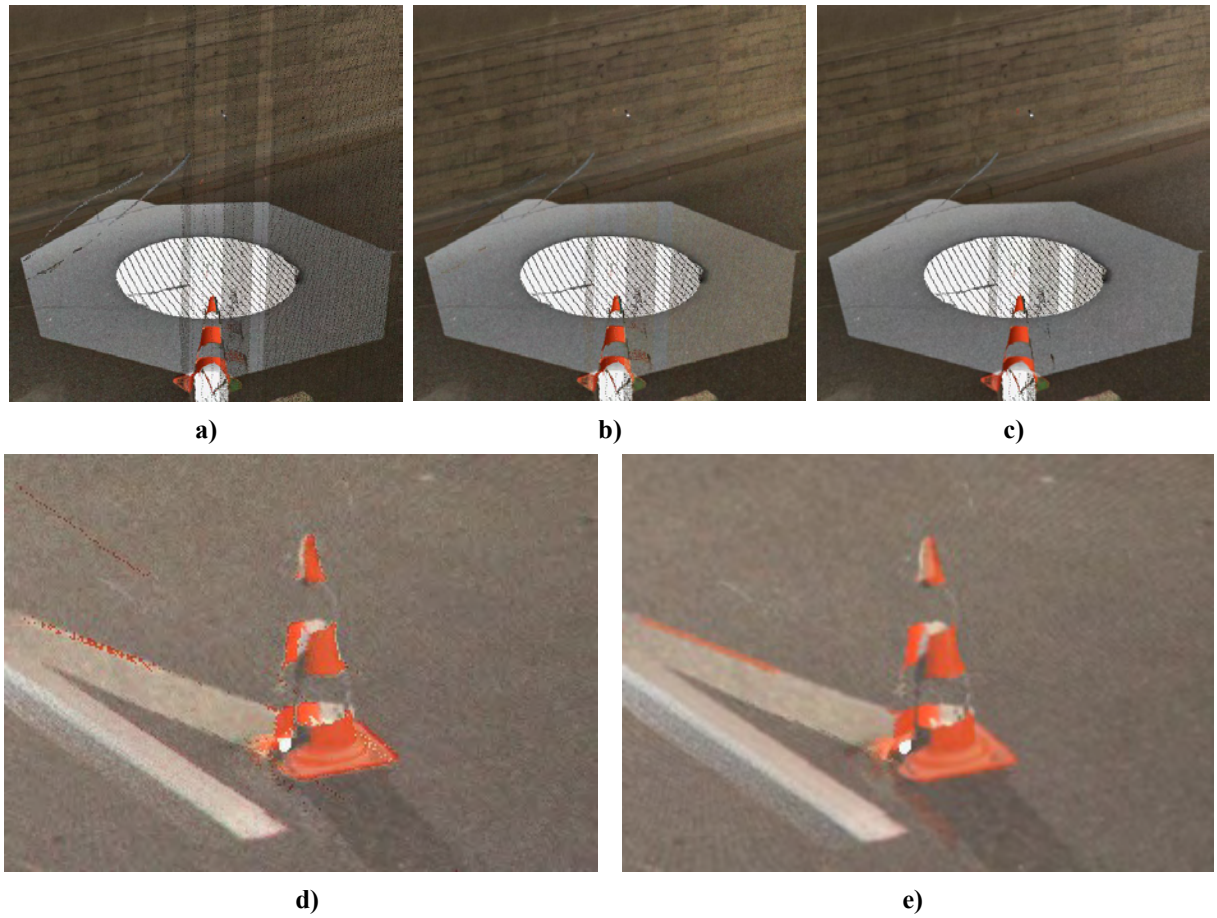


Figure 4. (a) - (c) Comparison of averaging in HSV colorspace without (left) and with Gaussian neighborhood (middle) and RGB space with Gaussian neighborhood (right). A part of the wall is visible between the viewer and the street. While in HSV space the wall is only brightened up but still visible, in RGB space it is smoothed away. (d) - (e) HSV smoothing (left) preserves sharp corners and the structure of the road in the rendering while using RGB (right) produces an antialiased image. In both cases the 3×3 neighborhood has been used.

averaging per fragment while not overblurring the image.

The advantage of using the HSV space lies in the error tolerance when applying the Gaussian. While RGB averaging produces smoother transitions in the image, it tends to blur away details in the rendering. HSV averaging in contrast preserves even small features and sharp edges if this is desired (Figure 4).

The amount of points necessary to successfully use this approach can be roughly approximated by the following calculation: Assume that a spherical volume element with a volume of $1\,\mathrm{cm}^3$ is projected to the center of a Full HD screen. Then the volume element gets projected to a spherical disc with the absolute (world-space) radius $r = \sqrt[3]{\frac{3}{4\pi}}$. Now let α be the vertical field of view of the virtual camera. The projected area of the volume element in distance d from the camera is then given by

$$A = \pi \cdot \left(\frac{1080\, r}{2 d \tan(\alpha)} \right)^2 .$$

Our experiments have shown that a number of seven to ten points per pixel usually suffice to yield smooth transitions. Therefore, in order to view a model from a given distance d, the resolution k of the dataset in terms of points per cm^3 has to satisfy $k \geq 10 \cdot A$, which corresponds to around 50 points per cm^3 for a view from a distance of 5 meters with $\alpha = 35°$. If this requirement is met the angle from which the point cloud is viewed is not of great importance since the geometric measuring accuracy is usually very high, producing only minor inconsistencies in the point-to-pixel mapping in adjacent views. It can only pose problems in the immediate vicinity of edges, when the depth threshold is so high that occluded points are included during blending. This, however, can be mitigated by a sufficiently low threshold and did not induce visible errors in our experiments.

Having normalized the luminance of our points we can fill remaining small holes in the rendering using the image-space filters proposed by Rosenthal and Linsen [RL08]. Finally we apply a simple triangulation scheme over a pixel's 8-connected neighborhood in image space to achieve a satisfying rendering enhanced with lighting and ambient occlusion (See Figure 8).

6. RESULTS AND DISCUSSION

We have tested our luminance correction approach in terms of quality and speed with the help of two types of datasets: unaltered real world data (Hinwil) and real world data with artificial noise (Lucy, dataset courtesy of the Stanford 3D Scanning Repository). We implemented the method using C++ and GLSL on an Intel Core i7-970 machine with an NVIDIA Quadro 4000 graphics card. All results were obtained by rendering to a viewport of 1920 × 1080 pixels.

The Hinwil dataset was acquired by scanning a bridge with several stationary high-resolution laser scanners. Two directly adjacent scans, each one containing about 30 million points, were registered using a geometry-based approach to obtain a dataset with 59 million points. The two scans were taken at different times of the day, resulting in different lighting conditions. An image-space rendering of the data is shown in Figure 6. It exhibits three common problems in such data that are highlighted and magnified:

1. Scattering: A part of the object was scanned from different scanners, but the point cloud from one scanner is significantly more dense in that region than the other's. The result is a noisy appearance of the surface because single points are scattered over the surface.
2. Shadowing: A part of the object can be seen from only one scanner, resulting in aliased lighting borders.
3. Border regions: These are the regions, where the point density of two scanners is nearly equal, causing sharp borders when applying a median filter.

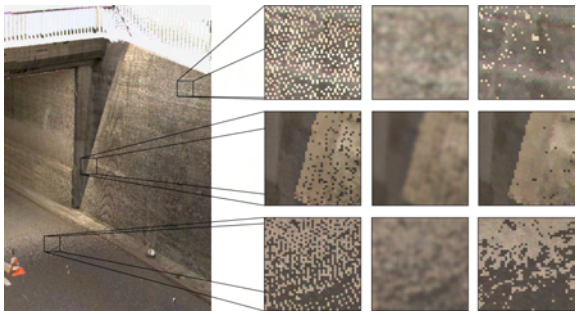
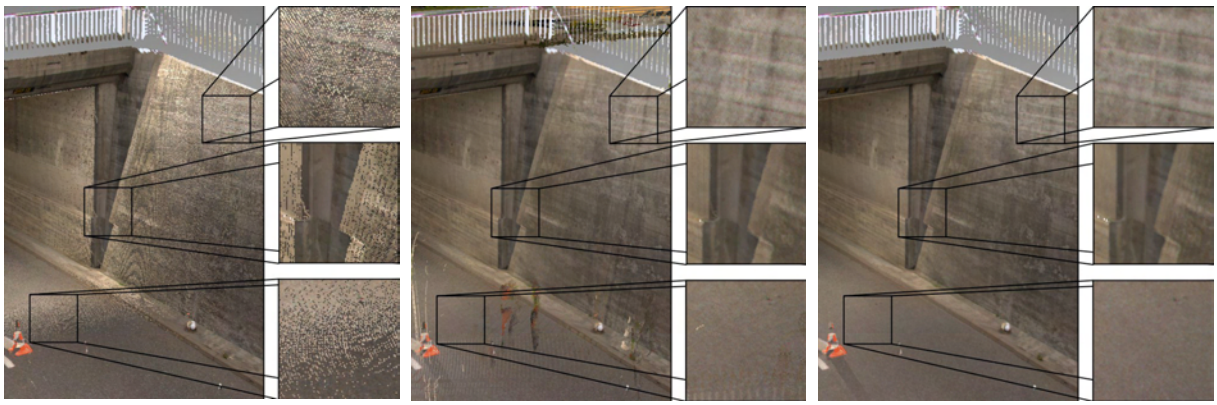


Figure 5. Image-space point-cloud rendering of the Hinwil dataset with closeup views of three problematic regions (left column). The application of standard image filters, like a Gaussian filter (middle column) or a median filter (right column) is not producing satisfying results.

The first problem can, in many cases, be satisfyingly handled by a median filter, which fails at the border regions since it produces a sharp border between the scans. Smoothing with a Gaussian filter yields slightly better results in border regions, but it keeps most of the scattered specks and leads to an overall blurry appearance of the rendered image, as illustrated in Figure 5. The shadowing problem is not easy to solve in image space, since we would have to correct the lighting over larger planar parts of the object.



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

Figure 6. Detail view of the Hinwil dataset using different approaches. For each approach, the overview and three closeup views are presented. (a) Image-space point-cloud rendering, exhibiting typical artifacts in regions with inconsistent lighting. (b) Image-space point-cloud rendering of the complete dataset with offline luminance correction (n=50). The preprocessing effectively eliminates the artifacts. (c) Image-space point-cloud rendering with image-space luminance correction. The visual quality is comparable to the offline method.

a) **b)**

Figure 7. An image-space rendering of the Lucy model with artificial noise (a) without and (b) with luminance correction. The model was artificially up-sampled to 40 million points to achieve a high-enough point density. The normalization was again carried out over the weighted neighborhood. In this image the prevailing noise is less visible than in the torus rendering, since the surface shape of the model is not as homogeneous as the torus.

# Points	Rendering	Rendering + Correction
1M	140 fps	80 fps
4M	75 fps	40 fps
16M	19 fps	10 fps
64M	5 fps	2.5 fps

Table 1. Performance of luminance correction. For different numbers of points the performance for just image-space point-cloud rendering and for the combination with image-space luminance correction is given in frames per second.

Our image-space approach eliminates most of these problems and even weakens the sharp shadow borders slightly. In Figure 6(c) a smooth transition from one scan to the other was achieved without emphasizing borders or blurring the image. To judge our approach in terms of quality, we compare the result to the one obtained by offline luminance correction, shown in Figure 6(b). Both approaches achieve similar results in terms of quality. However, the image-space luminance correction is able to interactively display the dataset without time-consuming preprocessing (offline luminance correction took more than one hour of computation time).

To evaluate the performance of our image-space algorithm we used the full Hinwil dataset with 138 million surface points and downsampled it to different resolutions. As shown in Table 1 the average rendering performance decreases by around 45% when applying image-space luminance correction. This is due to the two-pass nature of our approach.

As a real world dataset with artificial luminance noise we used the Lucy dataset, which consists of 14 million surface points. To achieve sufficient point density in close up views we upsampled the model to 40 million points and, since we were only interested in point data, we discarded the given connectivity information and colored each point uniformly white. We simulated the effects of scans under different lighting conditions by shading the surfaces points using the Phong model with a light source randomly positioned on a 90° circular arc directly above the model. An image-space rendering without luminance correction as well as an image-space rendering with

Figure 8. Rendering of a scene with image-space luminance correction, lighting and screen space ambient occlusion.

image-space luminance correction are shown in Figure 7. Our algorithm was able to largely remove the noise and yielded a satisfying rendering.

7. CONCLUSIONS

We have presented an approach for rendering point clouds with corrected luminance value at interactive frame rates. Our luminance correction operates in image space and is applied on the fly. As such, we have achieved our goal to allow for instantaneous rendering of large point clouds taken from multiple 3D scans of architecture. No preprocessing is necessary and the quality of the results is pleasing.

In order to work properly our algorithm relies on a sufficient number of points per pixel. Moreover, we observed in our experiments that single very bright scanlines from far away scanners were blended with darker regions of small point density, still leading to noticeable artifacts. This can be solved by considering only point clouds from directly adjacent scanners for blending which, at the present time, was done manually but will hopefully be automated in a future version.

8. ACKNOWLEDGMENTS

The authors would like to thank the enertec engineering AG (Winterthur, Switzerland) for providing us with the real-world data and for their close collaboration. This work was partially funded by EUREKA Eurostars (Project E!7001 "enercloud").

9. REFERENCES

[AA03] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In SMI '03: Proceedings of the Shape Modeling International 2003, pp.272–279, Washington, DC, USA, 2003. IEEE Computer Society.

[ABCO+03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. IEEE Transactions on Visualization and Computer Graphics, 9:3–15, 2003.

[AGDL09] Andrew Adams, Natasha Gelfand, Jennifer Dolson, and Marc Levoy. Gaussian kd-trees for fast high-dimensional filtering. ACM Transactions on Graphics, 28:21–33, 2009.

[BC04] Danny Barash and Dorin Comaniciu. A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift. Image and Vision Computing, 22(1):73 – 81, 2004.

[BJ10] Jongmin Baek and David E. Jacobs. Accelerating spatially varying gaussian filters. ACM Transactions on Graphics, 29:169–179, 2010.

[BR02] Fausto Bernardini and Holly Rushmeier. The 3d model acquisition pipeline. Computer Graphics Forum, 21(2):149–172, 2002.

[BWK02] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In Proceedings of the 13th Eurographics workshop on Rendering, pp.53–64, 2002.

[DRL10] Petar Dobrev, Paul Rosenthal, and Lars Linsen. Interactive image-space point cloud rendering with

transparency and shadows. In Vaclav Skala, editor, Communication Papers Proceedings of WSCG, The 18th International Conference on Computer Graphics, Visualization and Computer Vision, pp.101–108, Plzen, Czech Republic, 2010. UNION Agency – Science Press.

[GG07] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. ACM Transactions on Graphics, 26, 2007.

[GZPG10] P. Goswami, Y. Zhang, R. Pajarola, and E. Gobbetti. High quality interactive rendering of massive point models using multi-way kd-Trees. In Pacific Graphics Poster Papers, 2010.

[KK05] Hiroaki Kawata and Takashi Kanai. Direct point rendering on gpu. In George Bebis, Richard Boyle, Darko Koracin, and Bahram Parvin, editors, Advances in Visual Computing, volume 3804 of Lecture Notes in Computer Science, pp.587–594. Springer Berlin / Heidelberg, 2005.

[KS10] Michael Kass and Justin Solomon. Smoothed local histogram filters. ACM Transactions on Graphics, 29:100–110, 2010.

[LMR07] Lars Linsen, Karsten Müller, and Paul Rosenthal. Splat-based ray tracing of point clouds. Journal of WSCG, 15:51–58, 2007.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, SIGGRAPH '00, pp.131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[MKC07] Ricardo Marroquim, Martin Kraus, and Paulo Roma Cavalcanti. Efficient point-based rendering using image reconstruction. In Proceedings of the Symposium on Point-Based Graphics, pp.101–108, 2007.

[PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp.335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[RD10] R. Richter and J. Döllner. Out-of-core real-time visualization of massive 3D point clouds. In Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, pp.121–128, 2010.

[RL00] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp.343–352, 2000.

[RL08] Paul Rosenthal and Lars Linsen. Image-space point cloud rendering. In Proceedings of Computer Graphics International, pp.136–143, 2008.

[SMK07] R. Schnabel, S. Moeser, and R. Klein. A parallelly decodeable compression scheme for efficient point-cloud rendering. In Proceedings of Symposium on Point-Based Graphics, pp.214–226, 2007.

[SSS06] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3D. ACM Transactions on Graphics, 25:835–846, 2006.

[TS08] Thorsten Thormählen and Hans-Peter Seidel. 3D-modeling by ortho-image generation from image sequences. ACM Transactions on Graphics, 27:86:1–86:5, 2008.

[WBB+08] Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling, and Hans-Peter Seidel. Processing and interactive editing of huge point clouds from 3D scanners. Computers & Graphics, 32(2):204–220, 2008.

[WWPS10] Z. Wang, L. Wang, Y. Peng, and I. . Shen. Edge-preserving based adaptive icc method for image diffusion. In Proceedings of the 3rd International Congress on Image and Signal Processing, CISP 2010, volume 4, pp.1638–1641, 2010.

Subpixel Reconstruction Antialiasing for Ray Tracing

Chiu, Y.-F
National Tsing Hua University, Taiwan
yfchiu@ibr.cs.nthu.edu.tw

Chen, Y.-C
National Tsing Hua University, Taiwan
louis@ibr.cs.nthu.edu.tw

Chang, C.-F
National Taiwan Normal University, Taiwan
chunfa@ntnu.edu.tw

Lee, R.-R
National Tsing Hua University, Taiwan
rrlee@cs.nthu.edu.tw

ABSTRACT

We introduce a practical antialiasing approach for interactive ray tracing and path tracing. Our method is inspired by the Subpixel Reconstruction Antialiasing (SRAA) method which separates the shading from visibility and geometry sampling to produce antialiased images at reduced cost. While SRAA is designed for GPU-based deferred shading renderer, we extend the concept to ray-tracing based applications. We take a hybrid rendering approach in which we add a GPU rasterization step to produce the depth and normal buffers with subpixel resolution. By utilizing those extra buffers, we are able to produce antialiased ray traced images without incurring performance penalty of tracing additional primary rays. Furthermore, we go beyond the primary rays and achieve antialiasing for shadow rays and reflective rays as well.

Keywords: antialiasing, ray tracing, path tracing.

1 INTRODUCTION

With the abundance of computation power and parallelism in multicore microprocessors (CPU) and graphics processors (GPU), achieving interactive photorealistic rendering on personal computers is no longer a fantasy. Recently, we have seen the demonstration of real-time ray tracing [6, 17] and the emergence of real-time path tracing with sophisticated global illumination [2, 20]. Though real-time path tracing can produce rendering of photorealistic quality that include complex lighting effects such as indirect lighting and soft shadow, the illusion of a photograph-like image breaks down quickly when jaggy edges are visible (Figure 1 shows an example).

Jaggy edges are one of the typical aliasing artifacts in computer generated images. A straightforward antialiasing technique is to increase the sampling rate by taking multiple samples uniformly at various subpixel positions. However this approach induces significant performance penalty that makes it an afterthought in real-time ray tracing. A more practical approach is to increase subpixel samples adaptively for image pixels where discontinuity is detected. Although adaptive sampling approach avoids the huge performance hit of the multisampling approach, it still requires additional subpixel samples and introduces large variation to the estimation of rendering time.

In this work, we introduce an antialiasing approach that works well for real-time ray tracing and path tracing. We take a hybrid rendering approach in which we add a GPU rasterization step to produce the depth and normal buffers with subpixel resolution. By utilizing those extra buffers, we are able to produce antialiased ray traced images without incurring performance penalty of tracing additional primary rays. Our method is inspired by the Subpixel Reconstruction Antialiasing (SRAA) [3] which combines per-pixel shading with subpixel visibility to produce antialiased images. While SRAA is designed for GPU-based deferred shading renderer, we extend the concept to ray-tracing based applications. Furthermore, we apply our antialiasing approach to shadow and reflection which SRAA cannot resolve with its subpixel buffers.

Our main contributions in this work are:

- We propose an efficient antialiasing technique which improves the perception of photorealism in interactive or real-time ray tracing without sacrificing its performance.
- Unlike adaptive sampling or subpixel sampling, our approach does not penalize the performance of a CPU ray tracer because no additional primary ray needs to be traced. Our hybrid rendering approach obtains the necessary subpixel geometric information by leveraging the GPU rasterization pipeline.
- While SRAA works well for improving the sampling on image plane, we extend its application beyond the primary rays and achieve antialiasing for shadow rays and reflective rays as well.

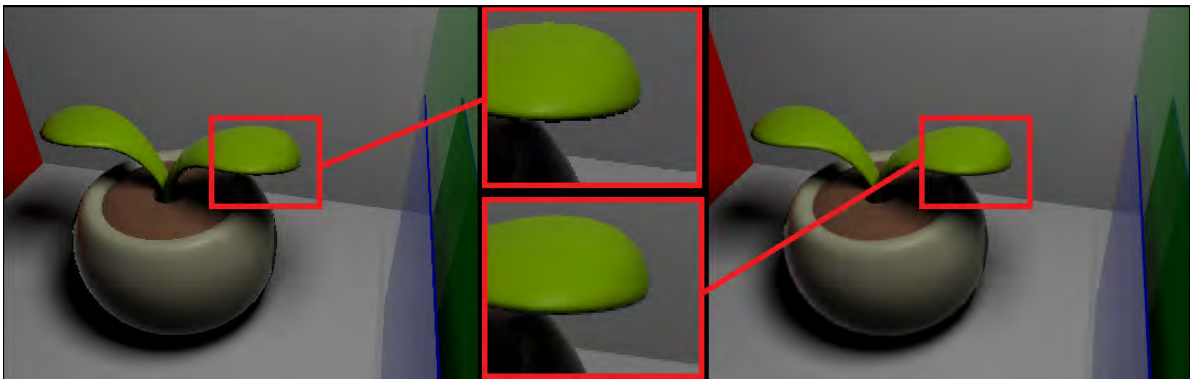


Figure 1: Antialiasing can improve the rendering quality in interactive ray tracing. The left image is rendered without applying any antialiasing method. The right image is rendered with our method using relatively inexpensive geometric information to improve expensive shading results.

2 RELATED WORK

2.1 Real-time Ray Tracing

With the rapid improvement of computation power and parallelism in multicore microprocessors (CPU) and graphics processors (GPU), various works have been published on speeding up the ray tracing, either on CPUs [1, 19], GPUs [5], or special-purpose platforms [21]. Recently, we have seen the demonstration of real-time ray tracing with Whitted-style reflection and refraction (a.k.a. specular rays) [6, 17] and the emergence of real-time path tracing with sophisticated global illumination [2, 20]. The NVIDIA OptiX acceleration engine [12] elevates rendering applications to a new level of interactive realism by greatly increasing ray tracing speeds with GPU solutions. While real-time ray tracing is now feasible, most renderers still rely on Monte Carlo path tracing to obtain more sophisticated global illumination effects such as soft shadow and indirect lighting. Noisy preview images are usually produced first at interactive rates and then gradually converge to high quality images. Therefore, antialiasing often becomes an afterthought as it further slows down the rendering.

2.2 Adaptive Sampling

The ray tracing algorithm is basically a loop over all screen pixels to find the nearest visible object in the scene. We can consider ray tracing as a point sampling based rendering method in signal processing view. However, point sampling makes an all-or-nothing choice in each pixel and thus leads to jaggies. Antialiasing of ray-traced images could be achieved by supersampling the image. However the supersampling approach demands significantly larger amount of computation resource. Therefore antialiasing by supersampling is rarely adopted by software renderers. Adaptive sampling [10, 11] reduces the overhead by casting additional rays only if significant color variation across image samples is detected. Variations of the adaptive sampling techniques have also been proposed in [8].

2.3 Post Filter Antialiasing

A small disadvantage of adaptive sampling is that some image pixels still need additional subpixel samples to be fully shaded or traced. It would be desirable if expensive shading could be avoided at additional subpixel locations. Reshetov [16] proposes an image filtering approach, Morphological antialiasing (MLAA) to recover edges from input image with per-pixel color information. However, this sort of color-only information could fail to identify some geometry edges, especially those edges without high contrast. Geometric Post-process Anti-Aliasing (GPAA) [13] and Geometry Buffer Anti-Aliasing (GBAA) [14] extend the MLAA ideas and use extra edge information explicitly to eliminate the jaggy edges. Normal Filter Anti-Aliasing (NFAA) [18] reduces aliasing by searching for contrasting luminosity changes in the final rendering image. It builds a normal displacement map to apply a per-pixel blur filter in highly contrast aliased areas. However, it softens the image due to the filtering of textures. More filter-based approaches are discussed in [7].

2.4 Shading Reconstruction Filter

Decoupled sampling [9, 15] presents an approach to generate shading and visibility samples at different rates in GPU pipelines to speed up the rendering in applications with stochastic supersampling, depth of field, and motion blur. Yang et al. [22] present a geometry-aware framebuffer level of detail (LOD) approach for controlling the pixel workload by rendering a subsampled image and using edge-preserving upsampling to the final resolution. Subpixel Reconstruction Antialiasing (SRAA) [3] takes a similar decoupled sampling approach and applies a cross-bilateral filter (as in the geometry-aware framebuffer LOD method) to

upscale shading information using subpixel geometric information that is obtained from the GPU rasterization pipeline. It is based on the assumption that the subpixel geometric information could be obtained much more easily without fully going through the expensive shading stage. SRAA can produce good edge antialiasing but it cannot resolve shading edges in texture, shadow, reflection and refraction. Our work follows the same assumption by avoiding emitting subpixel samples for the primary rays. This maintains the advantage over adaptive sampling because no subpixel ray needs to be traced.

3 ANTIALIASING

SRAA [3] relies on the fact that shading often changes more slowly than geometry in screen space and generates shading and visibility at different rates. SRAA performs high-quality antialiasing in a deferred rendering framework by sampling geometry at higher resolution than the shaded pixels. It makes three modifications to a standard rendering pipeline. First, it must produce normal and depth information at subpixel resolution. Second, it needs to reconstruct the shading values of sampled geometric subpixel from neighboring shaded samples with bilateral filter using the subpixel geometric (normal and depth) information. Finally, the subpixel shading values are filtered into an antialiased screen-resolution image.

SRAA detects the geometric edges with geometric information to resolve aliasing problem. However, the edges of shadow and reflection/refraction could not be detected by the subpixel geometric information generated from the eye position. For example, the shadow edges mostly fall on other continuous surfaces that have slowly changing subpixel depths and normals. To extend the SRAA concept to ray-tracing based applications, we perform antialiasing separately for primary rays, shadow rays and secondary rays to resolve this issue. The following subsections offer the detail.

3.1 Primary Ray

Like SRAA, our goal is to avoid the performance penalty of shading subpixel samples. In Figure 2, geometric information and shading are generated at different rates. Each pixel has 4 geometric samples on a 4×4 grid and one of those geometric samples is also a shaded sample. The shading value at each geometric sample is reconstructed by interpolating all shaded neighbors in a fixed radius using the bilateral weights. We take both depth and normal change into account when compute the bilateral weight. A neighboring sample with significantly different geometry is probably across a geometric edge and hence receives a low weight.

$$w_{ij} = G(\sigma_z(z_j - z_i))G(\sigma_n(1 - sat(n_j \cdot n_i))) \quad (1)$$

In Equation 1, $G(x)$ is the Gaussian function of the form $exp(-x^2)$. z_i and n_i are the depth and normal of the i^{th} subpixel sample. σ_z and σ_n are the scaling factors for controlling how quickly the weights fall off and allowing us to increase the importance of the bilateral filter. We set σ_z to 10 and σ_n to 0.25 in all our testing. The $sat(x)$ function is implemented as $max(0, min(1, x))$. The result w_{ij} is the weight associated with the j^{th} subpixel sample while performing shading reconstruction for the i^{th} subpixel sample.

For tracing the primary rays that are emitted from the eye position, we use a hybrid rendering approach that utilizes the GPU to generate the subpixel geometric information including position, normal and depth. We create 3 auxiliary geometric buffers to store position, normal and depth by GPU rasterization with the same resolution as the shaded buffer. Each geometric buffer is rendered with a subpixel offset applied to the projection matrix. The subpixel offset is applied not only to form a $4\times$ rotated-grid but also to do pixel alignment between rasterization and ray tracing rendering. Since the GPU rasterization pipeline produces the subpixel geometric information very efficiently, this overhead is insignificant when compared to the ray tracing stage.

3.2 Shadow Ray

As mentioned above in Section 3, the shadow edges cannot be detected by the geometric information that is generated from the eye position alone. What we need is subpixel information that is more meaningful to the shadow edges. The naive solution for shadow antialiasing is through a shadow map drawn at a higher resolution. However, this approach is inefficient because the increased resolution of the shadow map (from the light's view) does not contribute directly to the subpixels at the screen space. Therefore, we generate subpixel shadow information by ray casting and combine this shadow value with the bilateral filter weighting equation as shown in Equation 2. The subpixel shadow rays are generated by utilizing the position information in the geometric buffer as mentioned in Section 3.1.

Figure 2 shows our algorithm reconstructs the color value of a geometric sample in a non-shadowed area not only by taking the Euclidean distance and the normal change between the source and the target samples but also under the influence of shadow boundaries to exclude the neighboring samples in shadowed area. This is the reason why the original SRAA adds excessive blur to the shadow boundaries, yet our method achieves a better quality that is comparable to $16\times$ supersampling.

$$w_{ij} = \begin{cases} G(\sigma_z(z_j - z_i))G(\sigma_n(1 - sat(n_j \cdot n_i))), & \text{if } s_i = s_j \\ 0, & \text{if } s_i \neq s_j \end{cases} \quad (2)$$

In Equation 2, s_i is the shadow value of the i^{th} subpixel sample and is equal to 1 if it is in shadowed area. Otherwise it is equal to 0. If s_j is different from s_i, then the j^{th} subpixel falls on the other side of a shadow edge. Therefore we set the weight w_{ij} associated with the j^{th} subpixel to 0 to exclude it from the shading reconstruction for the i^{th} subpixel sample.

3.3 Secondary Ray

In the original SRAA framework, it uses geometric information to detect geometric edges in the subpixel reconstruction process. However, the edge of secondary shading (such as those from the reflection) cannot be detected by this geometric information generated from the eye position. Take the reflection rays for an example as shown in Figure 5 (c), if we perform subpixel shading reconstruction as shown in Equation 1 with the geometric information generated from the eye position, it will not be able to detect the edges of the reflected objects, and in consequence add excessive blur to the reflected colors.

Therefore we must take geometric information that is generated from the hit points of primary rays to perform subpixel-level bilateral filer when computing the shading value of the secondary rays that originate from the primary hit point. The subpixel secondary rays for hit points are generated by utilizing the position and normal information in the geometric buffer. Our method which performs subpixel reconstruction separately for primary and secondary shading achieves better quality than the original SRAA approach. Please see our results in Section 4 and Figure 5.

4 RESULT

Our algorithm is implemented using NVIDIA CUDA 4.0, the raytracer is built with OptiX 2.0 and rasterization with OpenGL. All results shown in this paper were obtained using an Intel Xeon E5504 processor and an NVIDIA Geforce GTX 570 GPU.

4.1 Quality

Figure 4 shows the quality comparison between our method and other antialiasing techniques in a Cornell box scene. The original SRAA adds excessive blur to the shadow, yet our method achieves similar quality to 16× supersampling.

Figure 5 highlights some interesting cases for primary shading, shadow and secondary shading in the Sponza scene. For the shading from primary rays, both our method and SRAA use the geometric information to improve image quality and the results are almost identical between ours and SRAA. For the shadow, our method uses both the geometry and the shadow edge information to perform subpixel reconstruction, thus produces better shadow line in the highlighted area than SRAA. For secondary shading, we perform subpixel reconstruction separately for primary and secondary shading, while SRAA uses only the final color of each sampled subpixel for this purpose. This results in over blurring for secondary shading in SRAA.

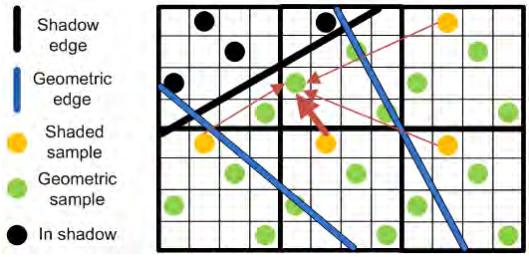


Figure 2: Here we add shadow edge into consideration to perform subpixel shading reconstruction. Each pixel has 4 geometric samples on a 4×4 grid. One of those geometric samples is also a shaded sample. Shading value for each geometric sample in non-shadowed area is reconstructed from nearby shaded samples except the shaded samples in shadowed area and weighted by their distance and the normal change between the source and the target sample.

To summarize, we observe that antialiasing with geometric information from primary rays could be problematic in some difficult cases and our method offers a solution to the highlighted cases in Figure 5.

Our method does have a limitation in handling material variation or textured surfaces. Figure 6 shows such an example where the floor contains patches of different colors. Since the extra subpixel depth and normal information does not help us detect the edges between patches of different colors, jagged edges could still appear on the floor.

4.2 Performance

There are two rendering passes in our current implementation. The first pass is the geometric information generation step and the second pass is the antialiasing process. Table 1 shows that the geometric information generation step with raytracer solution takes about 70 percent of the total processing time for rendering the Sponza scene [4] in Figure 5. This overhead to generate geometric information for primary rays can be reduced with a GPU hybrid solution. Figure 3 shows that our method maintains the interactive rate while rendering the Sponza scene in Figure 5 with a GPU hybrid solu-

	Rendering Pass		
Resolution	1^{st}	2^{nd}	Total
256x256	18	5	23
512x512	35	14	49
768x768	69	28	97
1024x1024	116	49	165

unit: millisecond

Table 1: Time measurement of our method for rendering the Sponza scene in Figure 5.The first pass is geometric information generation and the second pass is antialiasing process. Note that the time shown in first pass is measured with raytracer solution.

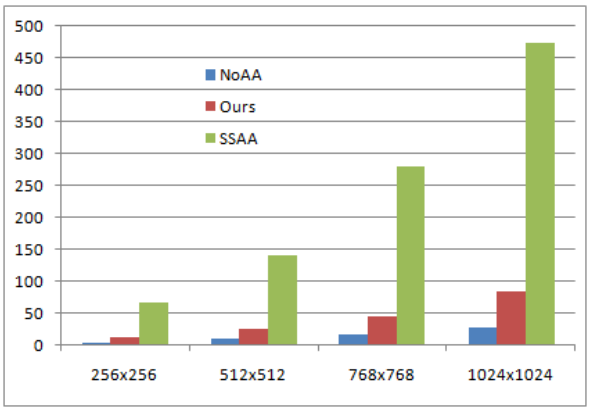


Figure 3: Performance comparison between NoAA (no antialiasing applied), our method with GPU hybrid approach, and SSAA (16× supersampling antialiasing) for rendering the Sponza scene under various output resolutions. The vertical axis is the rendering time in millisecond. The overall rendering performance of our method with a GPU hybrid approach is about 6× speedup in average compared to the 16× supersampling approach.

tion and achieves about 6× speedup in average compared to the 16× supersampling approach.

5 CONCLUSION

We introduce the concept in SRAA to path-tracing based rendering methods for antialiasing. Our method extends the subpixel geometric sampling concept beyond the primary rays and achieves antialiasing for shadow rays and reflective rays as well. By adopting a hybrid approach, our method improves the image quality without incurring performance penalty of tracing additional primary rays. We hope our method encourages the adoption of antialiasing even for the computationally constrained real-time ray tracing or path tracing.

6 ACKNOWLEDGEMENTS

This work is supported in part under the "Embedded software and living service platform and technology development project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs (Taiwan), and by National Science Council (Taiwan) under grant NSC 100-2219-E-003-002.

7 REFERENCES

[1] Carsten Benthin. *Realtime Ray Tracing on Current CPU Architectures*. PhD thesis, Saarland University, 2006.

[2] Jacco Bikker. Arauna real-time ray tracer and Brigade real-time path tracer.

[3] Matthäus G. Chajdas, Morgan McGuire, and David Luebke. Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 15–22, 2011.

[4] Marko Dabrovic. Sponza atrium, http://hdri.cgtechniques.com/ sponza/files/, 2002.

[5] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Realtime ray tracing on gpu with bvh-based packet traversal. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 113–118, 2007.

[6] Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. Interactive k-d tree gpu raytracing. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 167–174, 2007.

[7] Jorge Jimenez, Diego Gutierrez, Jason Yang, Alexander Reshetov, Pete Demoreuille, Tobias Berghoff, Cedric Perthuis, Henry Yu, Morgan McGuire, Timothy Lottes, Hugh Malan, Emil Persson, Dmitry Andreev, and Tiago Sousa. Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses*, 2011.

[8] Bongjun Jin, Insung Ihm, Byungjoon Chang, Chanmin Park, Wonjong Lee, and Seokyoon Jung. Selective and adaptive supersampling for real-time ray tracing. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 117–125, 2009.

[9] Gábor Liktor and Carsten Dachsbacher. Decoupled deferred shading for hardware rasterization. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 143–150, New York, NY, USA, 2012. ACM.

[10] Don P. Mitchell. Generating antialiased images at low sampling densities. In *Proceedings of the 14th annual conference on Computer graph-*

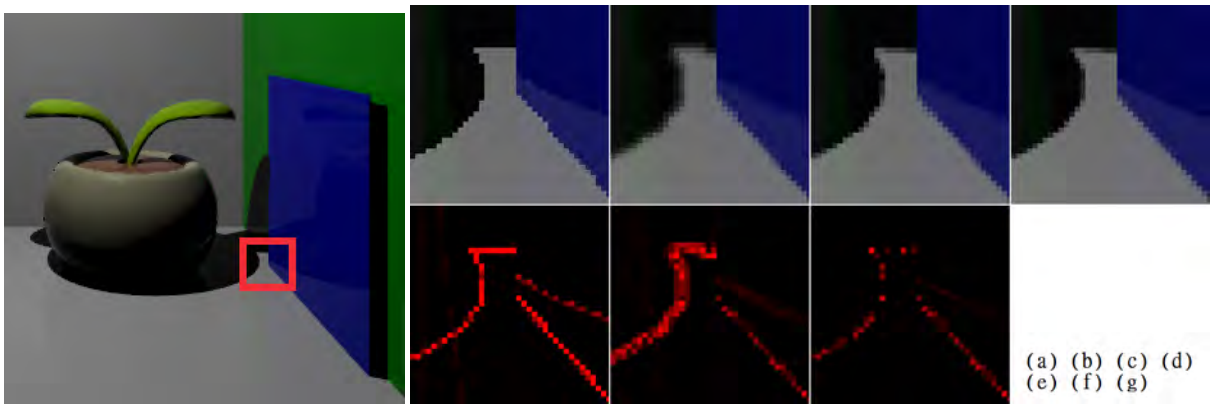


Figure 4: The leftmost image shows the Cornell box generated with our method. The smaller images to its right show the 8× zoom-in of the marked region under various antialiasing techniques. (a) is the result without any antialiasing. (b) is from SRAA. (c) is from our method. (d) is the reference image (16× supersampling antialiasing). (e)(f)(g) show the difference between (a)(b)(c) and the reference image (d) respectively. The original SRAA often adds excessive blur to the shadow and secondary shading, yet our method achieves similar quality to 16× supersampling.

ics and interactive techniques, SIGGRAPH '87, pages 65–72, 1987.

[11] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '89, pages 281–288, 1989.

[12] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. *ACM Trans. Graph.*, 29:66:1–66:13, July 2010.

[13] Emil "Humus" Persson. Geometric post-process anti-aliasing (GPAA), march 2011, http://www.humus.name/index.php?page=3d&id=86.

[14] Emil "Humus" Persson. Geometry buffer anti-aliasing (GBAA), july 2011, http://www.humus.name/index.php?page=3d&id=87.

[15] Jonathan Ragan-Kelley, Jaakko Lehtinen, Jiawen Chen, Michael Doggett, and Frédo Durand. Decoupled sampling for graphics pipelines. *ACM Trans. Graph.*, 30(3):17:1–17:17, May 2011.

[16] Alexander Reshetov. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 109–116, 2009.

[17] Min Shih, Yung-Feng Chiu, Ying-Chieh Chen, and Chun-Fa Chang. Real-time ray tracing with CUDA. In *Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing*, ICA3PP '09, pages 327–337, 2009.

[18] Styves. Normal filter anti-aliasing, http://www.gamedev.net/topic/580517-nfaa—a-post-process-anti-aliasing-filter-results-implementation-details, 2010.

[19] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004.

[20] Sven Woop and Manfred Ernst. Embree - photo-realistic ray tracing kernels, http://software.intel.com/en-us/articles/embree-highly-optimized-visibility-algorithms-for-monte-carlo-ray-tracing/, June 2011.

[21] Sven Woop, Jörg Schmittler, and Philipp Slusallek. RPU: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph.*, 24:434–444, July 2005.

[22] Lei Yang, Pedro V. Sander, and Jason Lawrence. Geometry-aware framebuffer level of detail. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering 2008)*, 27(4):1183–1188, 2008.

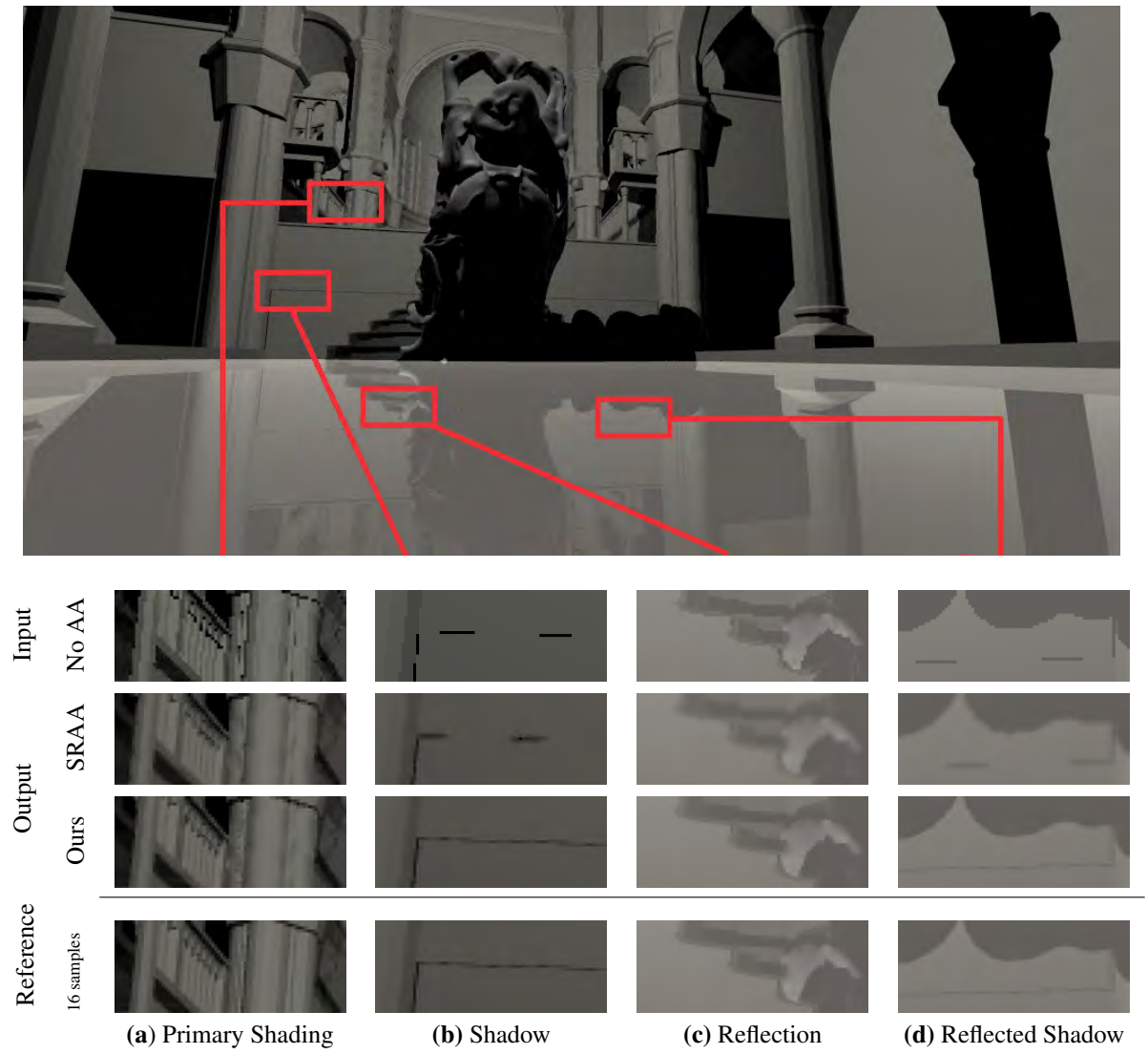


Figure 5: Quality comparison between our method and the other antialiasing techniques in highlighted areas of primary shading, shadow, reflection, and reflected shadow. (Row 1) No antialiasing, (Row 2) SRAA: one subpixel with shading value and 4 subpixels with primary geometric information, (Row 3) Ours: one subpixel with shading value and 4 subpixels with geometric information for primary, shadow and secondary rays, (Row 4) Reference image: 16× supersampling.

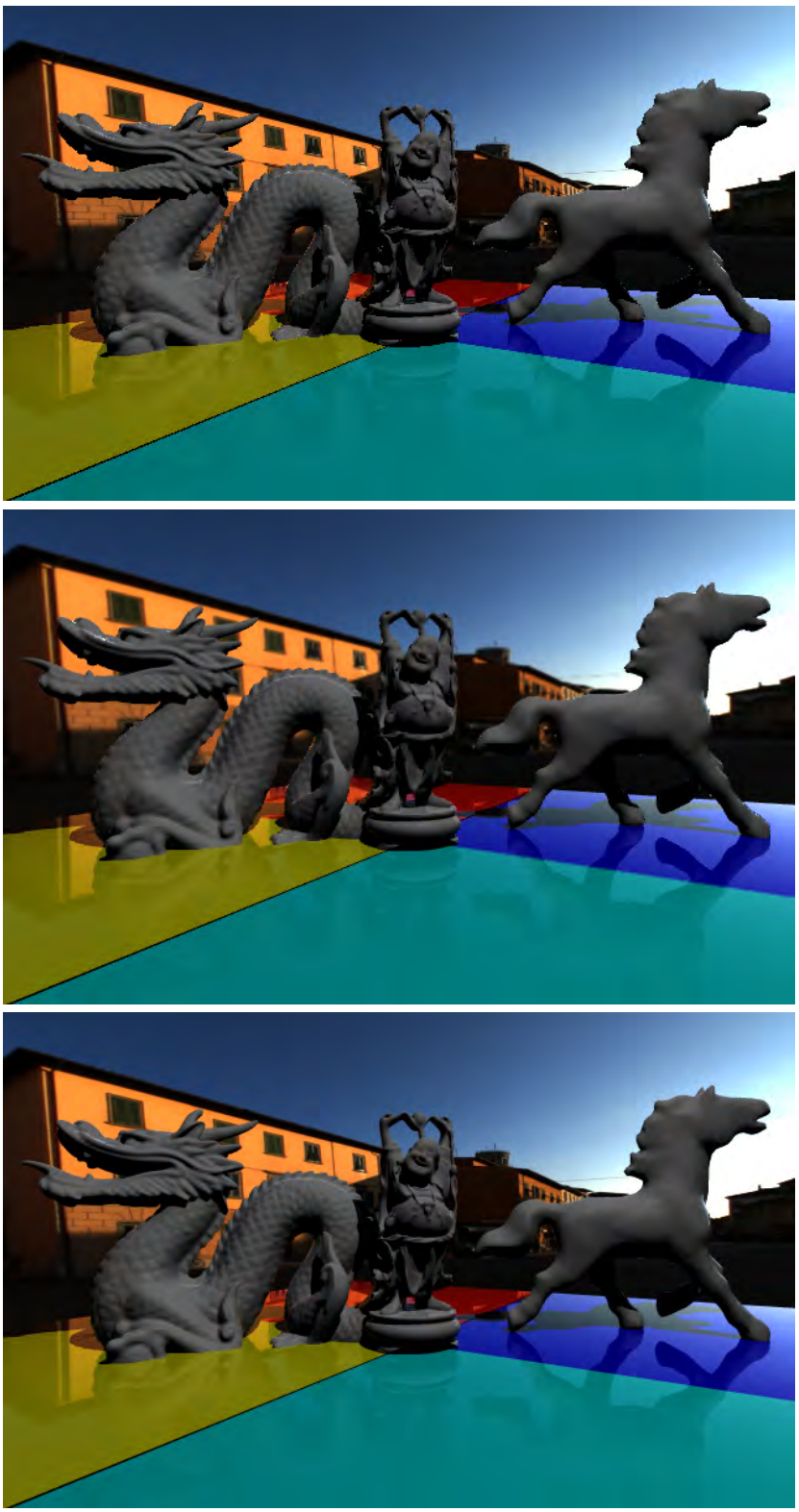


Figure 6: The scene in this figure shows a limitation of our method in handling material variation or textured surfaces. The floor contains patches of different colors. Since the extra subpixel depth and normal information does not help us detect the edges between patches of different colors, jagged edges still appear on the floor in the middle image that is rendered by our antialiasing method. For comparison, the top image shows the result without antialiasing and the bottom image is produced with 16× supersampling.

Elastically Deformable Models based on the Finite Element Method Accelerated on Graphics Hardware using CUDA

Mickeal Verschoor
Eindhoven University of Technology,
The Netherlands
m.verschoor@tue.nl

Andrei C. Jalba
Eindhoven University of
Technology, The Netherlands
a.c.jalba@tue.nl

Abstract

Elastically deformable models have found applications in various areas ranging from mechanical sciences and engineering to computer graphics. The method of Finite Elements has been the tool of choice for solving the underlying PDE, when accuracy and stability of the computations are more important than, e.g., computation time. In this paper we show that the computations involved can be performed very efficiently on modern programmable GPUs, regarded as massively parallel co-processors through Nvidia's CUDA compute paradigm. The resulting global linear system is solved using a highly optimized Conjugate Gradient method. Since the structure of the global sparse matrix does not change during the simulation, its values are updated at each step using the efficient update method proposed in this paper. This allows our fully-fledged FEM-based simulator for elastically deformable models to run at interactive rates. Due to the efficient sparse-matrix update and Conjugate Gradient method, we show that its performance is on par with other state-of-the-art methods, based on e.g. multigrid methods.

Keywords: Elastically deformable models, Finite Elements, sparse-matrix update, GPU.

1 INTRODUCTION

Mathematical and physical modeling of elastically deformable models has been investigated for many years, especially within the fields of material and mechanical sciences, and engineering. In recent years, physically-based modeling has also emerged as an important approach to computer animation and graphics modeling. As nowadays graphics applications demand a growing degree of realism, this poses a number of challenges for the underlying real-time modeling and simulation algorithms. Whereas in engineering applications modeling of deformable objects should be as accurate as possible compared to their physical counterparts, in graphics applications computational efficiency and stability of the simulation have most often the highest priority.

The Finite Element Method (FEM) constitutes one of the most popular approaches in engineering applications which need to solve Partial Differential Equations (PDEs) at high accuracies on irregular grids [PH05]. Accordingly, the (elastically) deformable object is viewed as a continuous connected volume, and the laws of continuum mechanics provide the governing PDE, which is solved using FEM. Other popular methods are the Finite Difference Method (FDM) [TPBF87], the Finite Volume Method (FVM) [TBHF03] and the Boundary Element Method (BEM) [JP99] (see [NMK*06, GM97]). FDM is the easiest to implement, but as it needs regular spatial grids, it is difficult to approximate the boundary of an arbitrary object by a regular mesh. FVM [TBHF03] relies on a geometrical framework, making it more intuitive than FEM. However, it uses heuristics to define the strain tensor and to calculate the force emerging at each node. BEM performs the computations on the surface of the model, thus achieving substantial speedups as the size of the problem is proportional to the area of the model's boundary as opposed to its volume. However, this approach only works for objects whose interior is made of homogeneous material. Furthermore, topological changes are more difficult to handle than in FEM methods [NMK*06].

In this paper we present a fully-fledged FEM-based simulator for elastically-deformable models, running solely on GPU hardware. We show that the involved computations can be performed efficiently on modern programmable GPUs, regarded as massively parallel co-processors through Nvidia's CUDA compute paradigm. Our approach relies on the fast GPU Conjugate Gradient (CG) method of [VJ12] to solve the resulting linear system. Since the topology of the deformed mesh does not change during the simulation, the structure of the sparse-matrix describing the linear system is reused throughout the simulation. However, during the simulation, the matrix values have to be updated efficiently. To achieve this, we propose a method that updates the sparse-matrix entries respecting the ordering of the data, as required by the CG method of [VJ12], see Sect. 5.4. Thanks to the optimized CG method and the efficient sparse-matrix update procedure, we ob-

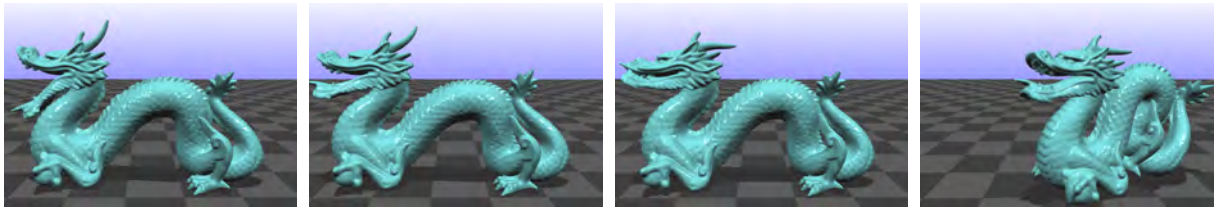


Figure 1: Effect of external (stretching) forces on an 'elastic' dragon.

tain results similar to state-of-the-art multigrid methods [DGW11].

The paper is organized as follows. Sections 3 and 4 describe the involved discretizations using FEM. Next, Section 5 presents the non-trivial parts of our GPU mapping, i.e., computing the local matrices, updating the global sparse matrix and solving the linear system. Finally, in Section 6 results are presented and analyzed.

2 PREVIOUS AND RELATED WORK

Bolz *et al.* [BFGS03], and Krüger and Westermann [KW03] were among the first to implement CG solvers on graphics hardware, using GPU programming based on (fragment) shaders. These authors had to deal with important limitations, *e.g.*, the lack of scatter operations, limited floating-point precision and slow texture switching based on pixel buffers, as exposed by the 'rendering-based' GPU-programming paradigm. One of the first GPU implementations of FEM is due to Rumpf and Strzodka [RS01], in the context of solving linear and anisotropic diffusion equations. Related work on GPU-accelerated FEM simulations also include the papers by Göddeke and collaborators [GST05, GST07, GSMY*07]. However, the emphasis is on improving the accuracy of *scientific* FEM-based simulations. Prior related work with respect to elastically deformable models, discretized using FEM, can be found in [HS04, MG04, ITF06]. They proposed methods which compensate for the rotation of the elements. Liu *et al.* [LJWD08] also present a FEM-based GPU implementation. Their results show that the involved CG method dominates the total computation time.

Since FEM often involves a CG solver, considerable research was done on efficiently mapping the CG method and Sparse Matrix-Vector Multiplications (SPMV) on modern GPUs using CUDA, see [BG08, BCL07, VJ12] and the references therein. Other approaches for solving the resulting PDE use multigrid methods, see e.g. [GW06]. An efficient GPU implementation of a multigrid method, used for deformable models, was recently presented in [DGW11]. Although multigrid methods typically converge faster than CG methods, implementing them efficiently on a GPU is a much more elaborate process. For example, invoking an iterative solver such as CG, constitutes only one of the steps of a multigrid method, the others being smoothing, interpolation and restriction.

3 ELASTICITY THROUGH THE METHOD OF FINITE ELEMENTS

As common in computer graphics applications (see [MG04] and the references therein), we employ a linearized model based on *linear* elasticity theory [PH05]. Further, to solve the underlying PDE we use the Method of Finite Elements with *linear tetrahedral elements*.

3.1 Continuum elasticity

In continuum elasticity, the deformation of a body, *i.e.*, a continuous connected subset M of $\mathbb{R}^3$, is given by the *displacement* vector field $\boldsymbol{u}(\boldsymbol{x}) = [u(\boldsymbol{x}), v(\boldsymbol{x}), w(\boldsymbol{x})]^T$, where $\boldsymbol{x} = [x, y, z]^T$ is some point of the body at rest. Thus, every point $\boldsymbol{x}$ of the undeformed body corresponds to a point $\boldsymbol{x} + \boldsymbol{u}(\boldsymbol{x})$ of the deformed one.

The equilibrium equation of the deformation is usually written in terms of the *stress tensor*, $\boldsymbol{\sigma}$. However, since it cannot be measured directly, one uses Cauchy's *linear strain tensor*, $\boldsymbol{\varepsilon}$, and some material parameters to approximate the stress inside the body. Similar to Hooke's law for a 1D spring, in 3D one has

$$\boldsymbol{\sigma} = \boldsymbol{D} \cdot \boldsymbol{\varepsilon}, \tag{1}$$

for each point of the body, where $\boldsymbol{D} \in \mathbb{R}^{6\times 6}$ is the so-called *material stiffness matrix* representing material parameters. The elastic force $\boldsymbol{f}_e$ acting at a point of the body is given by

$$\boldsymbol{f}_e = \boldsymbol{K} \cdot \boldsymbol{u} = \left(\boldsymbol{P}^T \boldsymbol{D} \boldsymbol{P}\right) \cdot \boldsymbol{u}, \tag{2}$$

with $\boldsymbol{K} \in \mathbb{R}^{3\times 3}$, $\boldsymbol{f}_e$ and $\boldsymbol{u} \in \mathbb{R}^{3\times 1}$. $\boldsymbol{K}$ represents the local *stiffness matrix* and $\boldsymbol{P} \in \mathbb{R}^{6\times 3}$ is a matrix of partial derivative operators.

3.2 System dynamics

Having defined the elastical forces acting in a body, we now derive the equations of motion required to simulate the dynamic behaviour of the object. The coordinate vectors $\boldsymbol{x}$ are now functions of time, *i.e.* $\boldsymbol{x}(t)$, such that the equation of motion becomes

$$m\ddot{\boldsymbol{x}} + c\dot{\boldsymbol{x}} + \boldsymbol{f}_e = \boldsymbol{F}_{ext}, \tag{3}$$

where m is the mass of a body particle at position $\boldsymbol{x}$, c the damping coefficient, $\boldsymbol{f}_e$ the elastic force and $\boldsymbol{F}_{ext}$ the vector of external forces, i.e., the gravitational force. We approximate Eq. (3) using a *semi-implicit* method, *i.e.*,

$$m\frac{\left(\boldsymbol{v}^{i+1}-\boldsymbol{v}^{i}\right)}{\Delta t}+c\boldsymbol{v}^{i+1}+\boldsymbol{K}\cdot\boldsymbol{u}^{i+i}=\boldsymbol{F}^{i}_{ext}. \quad (4)$$

$$\boldsymbol{x}^{i+1}=\boldsymbol{x}^{i}+\Delta t\,\boldsymbol{v}^{i+1}, \quad (5)$$

with $\boldsymbol{u}^{i+1}=\Delta t\boldsymbol{v}^{i+1}+\boldsymbol{x}^{i}-\boldsymbol{x}^{0}$, which can be rearranged as

$$\left(m+\Delta tc+\Delta t^{2}\boldsymbol{K}\right)\cdot\boldsymbol{v}^{i+1}= m\boldsymbol{v}^{i}-\Delta t\left(\boldsymbol{K}\cdot\boldsymbol{x}^{i}-\boldsymbol{K}\cdot\boldsymbol{x}^{0}-\boldsymbol{F}^{i}_{ext}\right). \quad (6)$$

3.3 Discretization using FEM

Within FEM, the continuous displacement field $\boldsymbol{u}$ is replaced by a discrete set of displacement vectors $\tilde{\boldsymbol{u}}$ defined only at the nodes of the elements. Within each element e the displacement field is approximated by

$$\boldsymbol{u}=\boldsymbol{\Phi}_e\cdot\tilde{\boldsymbol{u}}, \quad (7)$$

where $\boldsymbol{\Phi}_e\in\mathbb{R}^{3\times 12}$ is the matrix containing the element *shape functions* and $\tilde{\boldsymbol{u}}=[u_1,v_1,w_1,\ldots,u_4,v_4,w_4]^T$ is the vector of the nodal displacement approximations. Next, Galerkin's method of weighted residuals is applied over the whole volume V, in which the *weighting* functions are equal to the shape functions. Each term in Eq. (6) is weighted and approximated as in Eq. (7), which results in

$$\int_V \boldsymbol{\Phi}^T\left(m+\Delta tc+\Delta t^{2}\boldsymbol{K}\right)\boldsymbol{\Phi}\cdot\tilde{\boldsymbol{v}}^{i+1}dV= \int_V m\boldsymbol{\Phi}^T\boldsymbol{\Phi}\tilde{\boldsymbol{v}}^{i}dV- \Delta t\int_V\boldsymbol{\Phi}^T\left(\boldsymbol{K}\boldsymbol{\Phi}\cdot\tilde{\boldsymbol{x}}^{i}-\boldsymbol{K}\boldsymbol{\Phi}\cdot\tilde{\boldsymbol{x}}^{0}-\boldsymbol{\Phi}\cdot\tilde{\boldsymbol{F}}^{i}_{ext}\right)dV, \quad (8)$$

with $\boldsymbol{\Phi}^T$ the weighting functions. The equation above is defined for each individual element and generates one matrix consisting of the local mass ($\boldsymbol{M}_e$), damping ($\boldsymbol{C}_e$) and element stiffness ($\boldsymbol{K}_e$) matrices. Additionally, a local force matrix ($\boldsymbol{F}_e$) is generated, representing the net external force applied to the object. These local matrices are given by

$$\begin{aligned}\boldsymbol{M}_e&=\rho_e\textstyle\int_V\boldsymbol{\Phi}_e^T\boldsymbol{\Phi}_e\,dV\\ \boldsymbol{C}_e&=c\textstyle\int_V\boldsymbol{\Phi}_e^T\boldsymbol{\Phi}_e\,dV\\ \boldsymbol{K}_e&=\textstyle\int_V\boldsymbol{\Phi}^T\boldsymbol{P}^T\boldsymbol{D}\boldsymbol{P}\boldsymbol{\Phi}\,dV\\ \boldsymbol{F}_e&=\textstyle\int_V\boldsymbol{\Phi}_e^T\boldsymbol{\Phi}_e\cdot\tilde{\boldsymbol{F}}_{ext}\,dV,\end{aligned} \quad (9)$$

with ρ_e the density of element e. See [PH05] for more details on computing these matrices.

Finally, the global matrix $\boldsymbol{K}\in\mathbb{R}^{3n\times 3n}$ (with n the number of mesh vertices) is 'assembled' from individual element matrices. This resulting system is then solved using the *Conjugate Gradient* method for the unknown velocity $\boldsymbol{v}^{i+1}$, which is then used to update the positions of the nodes, see Eq. (5). Eq. (5) shows a first order method for updating the positions which can be replaced by higher order methods as described in [ITF06].

Unfortunately, the above equations for simulating elastic deformation only work fine as long as the model does not undergo *large rotations*. This is because linearized elastic forces are used, which are only 'valid' close to the initial configuration. Therefore we use the so-called *Element-based Stiffness Warping* or *Corotational Strain* method [MG04, HS04] to compensate for the rotation of the elements. To extract the rotation part of the deformation, we use the polar decomposition method proposed in [Hig86]. The rotation-free element stiffness matrix $\boldsymbol{K}_{re}$ then becomes $\boldsymbol{K}_{re}=\boldsymbol{R}_e\boldsymbol{K}_e\boldsymbol{R}_e^{-1}$, with $\boldsymbol{R}_e\in\mathbb{R}^{12\times 12}$ the rotation matrix for element e. Note that this gives rise to an initial elastic force $\boldsymbol{f}_{e0}=\boldsymbol{R}_e\boldsymbol{K}_e\cdot\boldsymbol{x}_0$, which replaces the term $\boldsymbol{K}\boldsymbol{\Phi}\cdot\tilde{\boldsymbol{x}}^0$ in the right-hand-side of Eq. (8).

4 OVERVIEW OF THE ALGORITHM

Algorithm 1 gives an overview of the simulation of elastically deformable models as described in Section 3. First, a tetrahedralization of the polygonal mesh representing the surface of the object is computed, see Section 5.5. Each tetrahedron is considered as an element in FEM. Then, the initial stiffness-matrices of the elements are computed (line 3); these matrices do not change during the simulation and thus are precomputed. Additionally, as the shape functions are constant during the simulation, we can pre-calculate most matrices from Eq. (9), using $\boldsymbol{N}_1=\boldsymbol{\Phi}_e^T\boldsymbol{\Phi}_e$. This matrix is identical for all elements and is therefore only computed once.

Algorithm 1 Simulation algorithm.

1:	Compute $\boldsymbol{N}_1$	see Eq. (9)
2:	**foreach** element e	
3:	Compute $\boldsymbol{K}_e$	see Eq. (9)
4:	**loop of the simulation**	
5:	**foreach** element e	
6:	Compute volume v_e	
7:	Compute $\boldsymbol{R}_e$	see Section 3.3
8:	Compute $\boldsymbol{K}_{re}=\boldsymbol{R}_e\boldsymbol{K}_e\boldsymbol{R}_e^{-1}v_e$	
9:	Compute $\boldsymbol{M}_e=\rho_e\boldsymbol{N}_1v_e$	
10:	Compute $\boldsymbol{C}_e=c\boldsymbol{N}_1v_e$	
11:	Compute $\boldsymbol{f}_{e0}=\boldsymbol{R}_e\boldsymbol{K}_e\cdot\boldsymbol{x}_0v_e$	
12:	Compute $\boldsymbol{F}_e=\boldsymbol{N}_1\cdot\tilde{\boldsymbol{F}}_{ext}v_e$	see Eq. (9)
13:	Compute $\boldsymbol{F}_{te}=\boldsymbol{M}_e\cdot\boldsymbol{v}^i-\Delta t\left(\boldsymbol{f}_{e0}-\boldsymbol{K}_{re}\cdot\boldsymbol{x}^i-\boldsymbol{F}_e\right)$	
14:	Compute $\boldsymbol{K}_{te}=\boldsymbol{M}_e+\Delta t\boldsymbol{C}_e+\Delta t^2\boldsymbol{K}_{re}$	
15:	Assemble global $\boldsymbol{K}$ and $\boldsymbol{F}$ using $\boldsymbol{K}_{te}$ and $\boldsymbol{F}_{te}$ of elements	
16:	Solve $\boldsymbol{K}\cdot\boldsymbol{v}^{i+1}=\boldsymbol{F}$ for $\boldsymbol{v}^{i+1}$	
17:	Update $\boldsymbol{x}^{i+1}=\boldsymbol{x}^i+\Delta t\boldsymbol{v}^{i+1}$	see Section 3.2

After all local matrices have been computed and stored (line 14), the global matrix is assembled

(line 15). The resulting linear system of equations is solved for velocities (line 16), which are then used to advance the position vectors (line 17).

5 GPU MAPPING USING CUDA

In this section we describe our GPU mapping of the simulation on NVida GeForce GPUs using CUDA [NVI]. First we shall give details about implementing the rotation extraction through polar decomposition. Then, we describe the computation of the local stiffness matrices which are used to assemble the global (sparse) stiffness matrix (matrix $\boldsymbol{K}$ in Algorithm 1). The resulting system of linear equations is solved using a Jacobi-Preconditioned CG Method.

5.1 Rotation extraction

As mentioned in subsection 3.3 we have to estimate the rotation of each element in order to calculate displacements properly. Finding the rotational part of the deformation matrix is done using a Polar Decomposition as described in [MG04, HS04, Hig86]. Although a large number of matrix inversions is involved, this can be done efficiently because small 4×4 matrices are used. Since each matrix contains 16 elements, we chose to map the computations of 16 such matrices to a single CUDA thread-block with 256 threads.

For computing the inverse of a 4×4 matrix we perform a *co-factor expansion*. Each thread within a thread-block computes one co-factor of the assigned matrix. Since the computation of a co-factor requires almost all values of the matrix, memory accesses have to be optimized. In order to prevent for possible bank-conflicts during the computation of the co-factors, each matrix is stored in one memory bank of shared memory. Accordingly, the shared-memory segment (of size 16×16 locations) is regarded as a matrix stored in row-major order, where each column represents a 4×4 local matrix. Therefore, each column (local matrix) maps exactly to one memory-bank. Since a large number of rotation matrices are computed in parallel, a large performance boost is obtained.

5.2 Local stiffness matrices

Solving a specific problem using FEM starts with describing the problem locally per element. Since a typical problem consists of a large number of elements, the computations involved per element can be easily parallelized. Further, since the matrices used to construct $\boldsymbol{K}_e$ are in $\mathbb{R}^{12\times12}$, we map the computation of each individual local element stiffness matrix to a thread-block containing 12×12 threads. The inner loop in Algorithm 1 is implemented using one or two CUDA kernels, depending on the architecture version. Instead of creating kernels for each individual matrix operation, we combine a number of them into one larger kernel. Since data from global memory can be reused multiple times, less global memory transactions are required, which improves the overall performance.

5.3 Solving the linear system

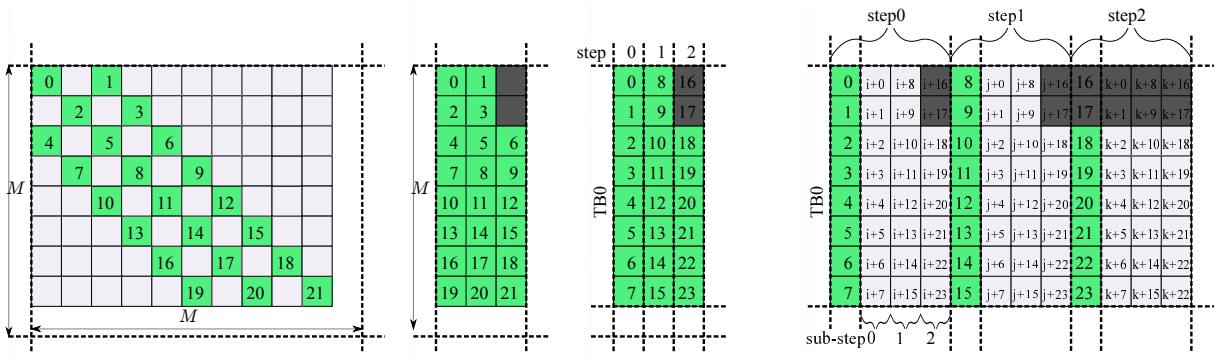
Given the local element matrices and load vectors, the global stiffness matrix of the system is assembled. Next, the system has to be solved for the unknown velocity $\boldsymbol{v}_{i+i}$. The (Jacobi-Preconditioned) CG method performs a large number of sparse matrix-vector multiplications and other vector-vector operations. Therefore, solving a large linear system efficiently, requires a fast and efficient implementation of sparse matrix-vector multiplications, which is highly-dependent on the layout used for storing the sparse matrix. Since three unknown values (components of the velocity vector) are associated to each mesh vertex, a block with 3×3 elements in the global matrix corresponds to each edge of the tetrahedral mesh. Therefore, a *Block-Compressed Sparse Row* (BCSR) format is very well suited for storing the global matrix, and thus improving the speed of the CG method.

Furthermore, since the vertex degree of internal nodes is constant in a regular tetrahedralization (see sect 5.5), the variation of the number of elements per row in the global matrix is minimal. Therefore, we use the optimized BCSR format from [VJ12]. This method efficiently stores a large sparse-matrix in BCSR format and reorders the blocks in memory to improve the efficiency of the memory transactions. This fact is very important since the main bottleneck of the CG method is the memory throughput. In [VJ12], through extensive experiments, it is shown that their optimized BCSR layout outperforms other storage formats for efficient matrix-vector multiplication on the GPU.

5.4 Global matrix update

Each local matrix represents a set of equations for each individual tetrahedron. To obtain the global system of equations, each component of each local matrix is added to the corresponding location of the global matrix. The location is obtained using the indices of the vertices for that specific element. Since the structure of the underlying mesh does not change during the simulation, also the structure of the global matrix remains unchanged. Therefore we assemble the global matrix only once and updates its values every time step. In this section, we propose an extension of [VJ12] which allows us to efficiently update a sparse matrix stored in the BCSR format.

For updating the global matrix, two approaches are possible. Within the first approach (*scatter*), all values of a local matrix are added to their corresponding values in the global matrix. When the local matrices are processed on the GPU, many of them are processed in parallel. Therefore, multiple threads *can* update the same



(a) Block layout of a sparse-matrix: Each green block stores $N \times N$ values and its position within the block-row. Numbers represent memory locations. Each gray block contains zero-values and is not explicitly stored. M represents the dimension of the matrix.

(b) BCSR layout: Each block-row is compressed; an additional array with indices to the first block in a block row is necessary (not shown here).

(c) Blocks of consecutive block rows are mapped to a thread block ($TB0$). Blocks mapped to the same thread block are reordered so that blocks processed in the *same step* are continuous in memory. Padding is required (gray blocks).

(d) Updating matrix blocks (green), requires the associated local values. The indices of these values are stored in *index blocks* (gray), in the same order as the matrix blocks. Within each sub-step, a set of continuous index-blocks are loaded and used to fetch the corresponding values from the local matrices. The dark-gray blocks are used for padding and contain -1's. i, j, k represent (starting) offsets in memory.

Figure 2: Updating the sparse matrix: the initial sparse matrix is created, stored and processed, (a), (b) and (c). Updating the matrix is done by collecting the corresponding values from the local matrices, (d).

value in the global matrix at the same time, which results in *race conditions*. In order to prevent race conditions from appearing, access to the values of the global matrix would have to be serialized.

The second approach is to *gather* per element in the global matrix, the corresponding values from the local matrices. To do so, the indices of all associated local values are stored per element in the global matrix. Each index represents the position of the local value in an array A, which stores the values of all local matrices. Given these indices per global element value, the local values are looked-up and used to update the corresponding value in the global matrix.

Within the optimized BCSR implementation of [VJ12], the global sparse-matrix is divided in $N \times N$-sized blocks, Fig. 2(a). Next, block rows are compressed and sorted by length, Fig. 2(b). Finally, a number of consecutive block rows are grouped and mapped to a CUDA thread block. Within each group of block rows, the blocks are reordered in memory, such that accessing these blocks is performed as optimal as possible. Accessing the blocks (for e.g. a multiplication) is done as follows. First, all threads of a thread-block ($TB0$) are used to access the blocks mapped to it in the first step (step 0), see Fig. 2(c). Each thread computes an index pointing to these blocks. Next, blocks $0-7$ are loaded from the global memory. Note that these are the same blocks appearing in the first column of Fig. 2(b). For the next step, each thread increases its current index, such that the next set of blocks $(8-15)$ can be loaded (step 1). Note that all block rows must have the same length, and therefore, empty blocks must be padded (blocks 16 and 17).

To actually update the data blocks of the global matrix, we use a *gather* approach. Accordingly, $N \times N$-sized *index blocks* are used for each matrix block, see Fig. 2(d). Since the matrix blocks have a specific ordering, the same ordering is used for the index-blocks. For each step, a number of *sub-steps* is performed. Within each sub-step a set of index-blocks is loaded from memory, given a start offset (i, j or k in Fig. 2(d)). Then, for each index-block, its $N \times N$ values (indices) are used to fetch the corresponding $N \times N$ data values from local matrices, stored in global memory. Please note that the $N \times N$ data values fetched using one $N \times N$ index-block, do not come, in general, from the same local matrices. To accumulate the local contributions, an array (stored in shared memory) is used. If an index has value -1, no update is performed. For the next sub-step, the indices pointing to the index blocks are increased. Therefore, per step, the number of index blocks for each processed matrix block must be equal, which requires padding with '-1' index blocks. The advantage of this approach is that loading the indices and writing the updated values always result in an optimal throughput. Loading the actual local-element values is in general not optimal.

5.5 Tetrahedralization and rendering

The quality of the tetrahedral mesh is essential for efficiently simulating a deforming elastic object represented by a polygonal mesh. We have experimented with tetrahedralizations in which the surface mesh forms the outer boundary of the tetrahedral mesh. Since the tri-

angles of the surface mesh can have a high variety in size, the generated tetrahedralization also contains tetrahedral elements with a high variation in size and configuration. This can have a negative effect on the quality of the tetrahedralization. Therefore, we chose to create a tetrahedral mesh, using equi-sized elements, which however, may result in a rather rough approximation of the original surface mesh. We tackle this problem by coupling the input polygonal mesh to the (deforming) tetrahedral mesh, as follows.

First, a regular 3D grid of N^3 voxels is created, in which each voxel containing a part of the surface is marked as important; typical values for N are $32, 64$ or 128. Next, a regular tetrahedralization of the grid is created using equi-sized tetrahedral elements, and each element containing at least one important vertex of the grid, is stored. Further, the inner volume of the object is tetrahedralized using the same equi-sized tetrahedral elements. Next, in order to reduce the amount of elements, those elements belonging to the inner volume are merged together into fewer larger ones. This reduces the total amount of elements and thus the total computation time. Note however that this approach is most useful with models which have large internal volumes, similar to the bunny in Figure 5. Finally, the original surface mesh is coupled with the tetrahedral one similar to [MG04]: each vertex in the original surface mesh is mapped to exactly one tetrahedron, and its barycentric coordinates in that tetrahedron are stored along with the vertex coordinates.

When the new positions of the tetrahedra are computed, the surface mesh is also updated. To compute new positions of the deformed surface mesh, for each vertex of the input mesh, the positions of the four vertices of the corresponding tetrahedron are looked-up and interpolated using the barycentric coordinates of the original vertex.

6 RESULTS

All experiments have been performed on a machine equipped with an Intel Q6600 quad-core processor and a GeForce GTX 570 with 1.2 Gb of memory.

Figure 3 shows the performances obtained for computing the local element matrices (*Matrix*), the rotation matrices (*Rotation*), solving the resulting linear system (*CG*), performing a single SpMV (*SpMV*), and the total performance (*Total*) as a function of the number of elements. *Steps/sec* is the corresponding number of simulation steps performed per second. Similarly, Figure 4 shows the computation time per simulation time-step. For each model, we have used the following material parameters: Young's modulus of elasticity, $E = 5 \times 10^5 N/m^2$; Poisson's ratio, $\mu = 0.2$; density, $\rho = 1000 KG/m^3$. Furthermore, the time step of the simulation $\Delta t = 0.001$ and the volume of each initial element $v_e = 1.65 \times 10^{-6} m^3$. Each model used in

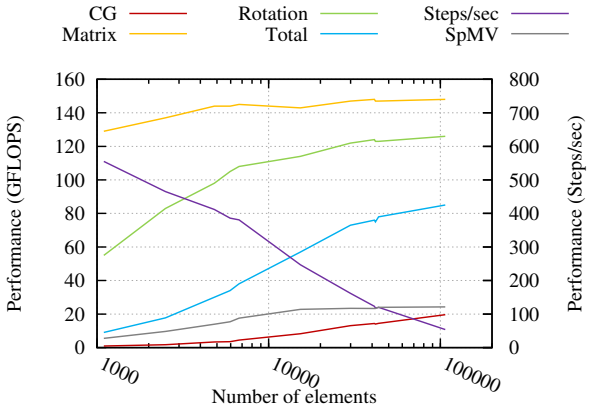


Figure 3: Performance results with different numbers of elements. *CG* represents the performance of the CG solver, *Matrix* – the performance for computing the local element matrices, *Rotation* – the performance of the rotation extraction procedure, *SpMV* – the performance of the SpMV operation; *Total* represents the overall performance. *Steps/sec* represents the number of simulation steps per second. The global-matrix update was performed with an effective throughput of 50 GB/sec.

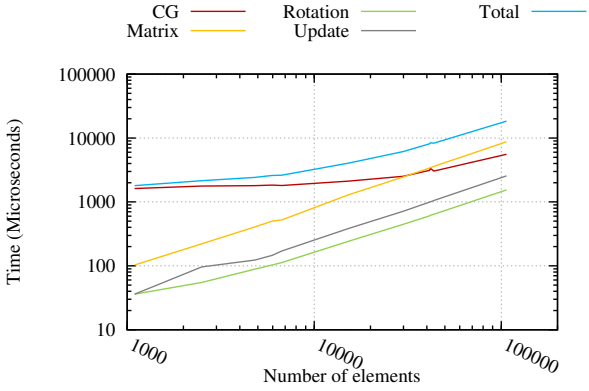


Figure 4: Timing results with different numbers of elements, per time step. *CG* represents the time of the CG solver, *Matrix* – the time for computing the local element matrices, *Rotation* – the time of the rotation extraction procedure; *Total* represents the total elapsed time per time-step.

this paper is scaled such that each dimension is at most 66 *cm* and is tetrahedralized as described in Section 5.5. With these settings, the CG solver found a solution for each model in 5 to 18 iterations. In order to obtain a generic performance picture, we have fixed the number of iterations to 18, which resulted in the performances from Fig. 3.

Within Figure 3 a number of interesting patterns can be seen. First, the performance for computing the local element matrices reaches its maximum very soon. Since each matrix is mapped to exactly one thread-

block, a large amount of thread-blocks is created, resulting in a 'constant' performance. Second, the performance figures for computing the rotation matrices show a larger variation. Since 16 rotation matrices are processed by one thread-block, a significantly smaller amount of thread-blocks is used. Finally, the performance of the CG method seems to be low compared to the other operations. The CG method operates on a global sparse-matrix and performs a large number of sparse-matrix vector multiplications (SPMVs) and vector-vector operations, for which the performances are mainly bound by the memory throughput. However, the CG performances from Figure 3 agree with those from [VJ12], given the dimensions of the problem.

The measured, effective throughput for updating the global matrix was about 50 GB/sec, in all cases with more than $5k$ elements. Since this operation transfers a large amount of data, the memory bus is saturated very soon, resulting in a good throughput. However, since not all transactions can be coalesced, the maximum throughput is not reached. This operation is very similar to an SPMV with 1×1 blocks, but now for a matrix containing $d\times$ more elements, with d the degree of internal nodes in the model. This observation shows that the measured throughput is close to the expected one, according to the results in [VJ12].

As expected, the total performance increases with the number of elements. This shows that the computational resources are used efficiently for larger models. The number of elements, for which the maximum performance is reached, depends on the actual GPU mapping of the computations. For example, the CG solver does not reach its maximum performance for $100k$ elements, while the computation of the local element matrices reaches its peak at $5k$ elements. Due to this, one can expect better performances for the CG method when larger models are used. Furthermore, for models having less than $30k$ elements, the total computation is dominated by the time spent by the CG solver. For larger models, more time is spent on computing the local matrices, see Figure 4.

The measured overall performance is based on the total time needed per simulation step, which includes all operations performed, except the rendering of the model. Figure 3 also shows the number of simulation steps performed per second, given the number of elements; these numbers are based on the total computation time. Accordingly, even for large models, interactive frame rates can be reached. A rough comparison of the obtained performance and frame rate with other state-of-the-art multigrid GPU implementations [DGW11] shows that, even if in theory the CG method converges slower than multigrid, comparable results *can* be obtained for similar models. We assume that memory transactions in our method are more efficient, despite of transferring more data. However, more research is required to get a full understanding of the differences between both methods performed on modern GPUs, with respect to performance figures. Finally, Figures 1, 5, 6, 7, 8 and 9 show example results from our simulations.

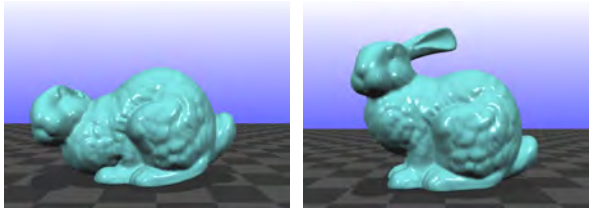


Figure 5: Material properties and collision handling. *Left*: flexible material ($E = 5 \times 10^4$). *Right*: stiffer material ($E = 5 \times 10^5$). Simulation rate: 120 frames per second.

Figure 6: Left: stretching and deforming a model using external forces. Right: deformation after releasing external forces. Simulation rate: 118 frames per second.

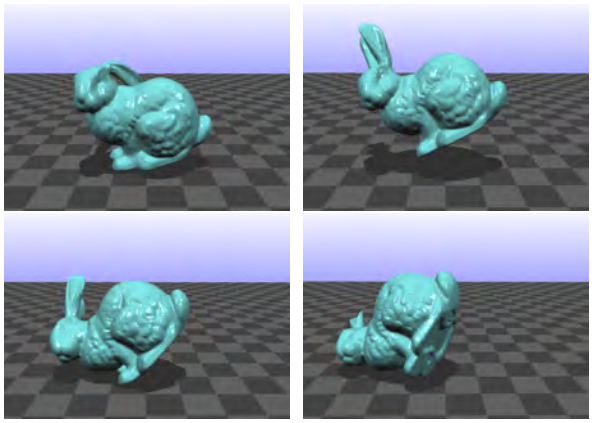


Figure 7: Bunny bouncing on the floor. Simulation rate: 120 frames per second.

7 CONCLUSIONS

We have presented an efficient method for simulating elastically deformable models for graphics applications, accelerated on modern GPUs using CUDA. Our method relies on a fast Conjugate Gradient solver and an efficient mapping of the SPMV operation on modern GPUs [VJ12]. Since the topology of the underlying grid

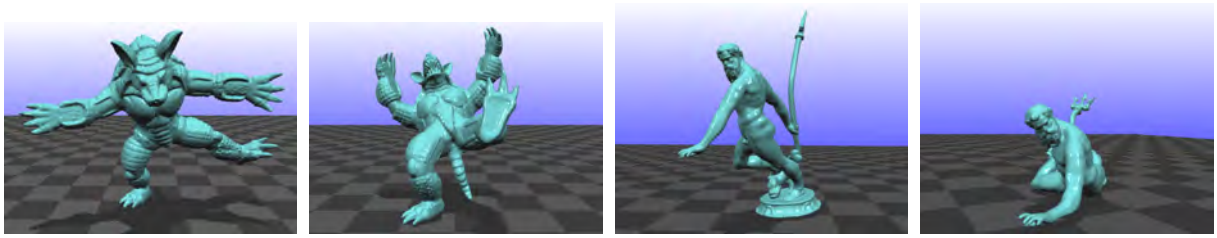


Figure 9: Other simulation results. Simulation rate: 160 frames per second.

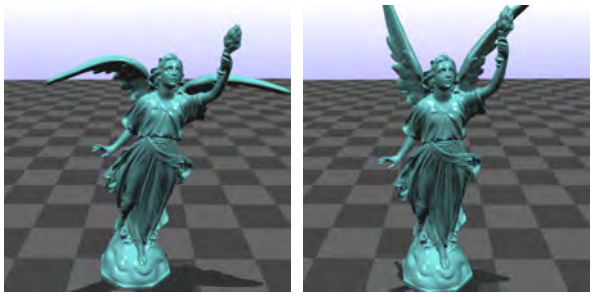


Figure 8: Left: applying external forces on the wings. Right: after releasing the external forces. Simulation rate: 116 frames per second.

does not change during the simulation, data structures are reused for higher efficiency. To further improve the performance, we proposed a scheme which allows to efficiently update the sparse matrix, during the simulation.

In future work we will investigate the performance of this method when multiple GPUs are used. Furthermore, we will investigate the performance difference between traditional CG methods and multigrid methods performed on modern GPUs. Also, we plan to enhance the simulation to allow for plastic behaviour as well as brittle and fracture of stiff materials.

REFERENCES

[BCL07] BUATOIS L., CAUMON G., LÉVY B.: Concurrent Number Cruncher: An Efficient Sparse Linear Solver on the GPU. In *High Perf. Comp. Conf. (HPCC)* (2007). 2

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *Proc. SIGGRAPH'03* (2003). 2

[BG08] BELL N., GARLAND M.: *Efficient Sparse Matrix-Vector Multiplication on CUDA*. Tech. Rep. NVR-2008-004, NVidia, 2008. 2

[DGW11] DICK C., GEORGII J., WESTERMANN R.: A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory 19*, 2 (2011), 801–816. 2, 7

[GM97] GIBSON S., MIRTICH B.: *A Survey of Deformable Modeling in Computer Graphics*. Tech. Rep. TR-97-19, MERL, Cambridge, MA, 1997. 1

[GSMY*07] GÖDDEKE D., STRZODKA R., MOHD-YUSOF J., MCCORMICK P., BUIJSSEN S. H., GRAJEWSKI M., TUREK S.: Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing 33*, 10–11 (2007), 685–699. 2

[GST05] GÖDDEKE D., STRZODKA R., TUREK S.: Accelerating double precision FEM simulations with GPUs. In *Proc. ASIM 2005 - 18th Symp. on Simul. Technique* (2005). 2

[GST07] GÖDDEKE D., STRZODKA R., TUREK S.: Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *Int. Journal of Parallel, Emergent and Distributed Systems 22*, 4 (2007), 221–256. 2

[GW06] GEORGII J., WESTERMANN R.: A multigrid framework for real-time simulation of deformable bodies. *Computers & Graphics 30*, 3 (2006), 408–415. 2

[Hig86] HIGHAM N. J.: Computing the polar decomposition – with applications. *SIAM Journal of Scientific and Statistical Computing 7* (1986), 1160–1174. 3, 4

[HS04] HAUTH M., STRASSER W.: Corotational simulation of deformable solids. In *WSCG* (2004), pp. 137–144. 2, 3, 4

[ITF06] IRVING G., TERAN J., FEDKIW R.: Tetrahedral and hexahedral invertible finite elements. *Graph. Models 68*, 2 (2006), 66–89. 2, 3

[JP99] JAMES D. L., PAI D. K.: ArtDefo: accurate real time deformable objects. In *Proc. SIGGRAPH'99* (1999), pp. 65–72.

1

[KW03] KRÜGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. In *Proc. SIGGRAPH'03* (2003), pp. 908–916. 2

[LJWD08] LIU Y., JIAO S., WU W., DE S.: Gpu accelerated fast fem deformation simulation. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on* (30 2008-dec. 3 2008), pp. 606 –609. 2

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proc. Graphics Interface 2004* (2004), pp. 239–246. 2, 3, 4, 6

[NMK*06] NEALEN A., MÜLLER M., KEISER R., BOXERMANN E., CARLSON M.: Physically based deformable models in computer graphics. *Computer Graphics Forum 25* (2006), 809–836. 1

[NVI] NVIDIA CORPORATION: *Compute Unified Device Architecture programming guide*. Available at `http://developer.nvidia.com/cuda`. 4

[PH05] PEPPER D. W., HEINRICH J. C.: *The Finite Element Method: Basic Concepts and Applications*. Taylor and Francis, 2005. 1, 2, 3

[RS01] RUMPF M., STRZODKA R.: Using graphics cards for quantized FEM computations. In *Proc. IASTED Vis., Imaging and Image Proc.* (2001), pp. 193–202. 2

[TBHF03] TERAN J., BLEMKER S., HING V. N. T., FEDKIW R.: Finite volume methods for the simulation of skeletal muscle. In *In SIGGRAPH/Eurographics symposium on Computer Animation* (2003), pp. 68–74. 1

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proc. SIGGRAPH'87* (1987), pp. 205–214. 1

[VJ12] VERSCHOOR M., JALBA A. C.: Analysis and performance estimation of the conjugate gradient method on multiple gpus. *Parallel Computing* (2012). (in press). 1, 2, 4, 5, 7

VISUALIZATION OF FLOW FIELDS IN THE WEB PLATFORM

Mauricio Aristizabal
Universidad EAFIT
[1]CAD/CAM/CAE Laboratory
Carrera 49 # 7 Sur - 50
Colombia (050022), Medellin, Antioquia
maristi7@eafit.edu.co

John Congote[1]
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
jcongote@vicomtech.org

Alvaro Segura
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
asegura@vicomtech.org

Aior Moreno
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
amoreno@vicomtech.org

Harbil Arregui
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
harregui@vicomtech.org

Oscar E. Ruiz
Universidad EAFIT
CAD/CAM/CAE Laboratory
Carrera 49 # 7 Sur - 50
Colombia (050022), Medellin, Antioquia
oruiz@eafit.edu.co

ABSTRACT

Visualization of vector fields plays an important role in research activities nowadays. Web applications allow a fast, multi-platform and multi-device access to data, which results in the need of optimized applications to be implemented in both high and low-performance devices. The computation of trajectories usually repeats calculations due to the fact that several points might lie over the same trajectory. This paper presents a new methodology to calculate point trajectories over a highly-dense and uniformly-distributed grid of points in which the trajectories are forced to lie over the points in the grid. Its advantages rely on a highly parallel computing implementation and in the reduction of the computational effort to calculate the stream paths since unnecessary calculations are avoided by reusing data through iterations. As case study, the visualization of oceanic streams in the web platform is presented and analyzed, using WebGL as the parallel computing architecture and the rendering engine.

Keywords

Streamlines, Trajectory, Hierarchical Integration, Flow Visualization, WebGL.

1 INTRODUCTION

Vector field visualization plays an important role in the automotive and aero-spatial industries, maritime transport, engineering activities and others. It allows the detection of particularities of the field such as vortexes or eddies in flow fields, but also permits exploring the entire field behavior, determining flow paths.

In particular, ocean flow visualization is important in maritime navigation and climate prediction, since the movement of sea water masses produces variations in air temperature and wind currents. Therefore, flow visualization becomes determinant to represent the ocean's behavior.

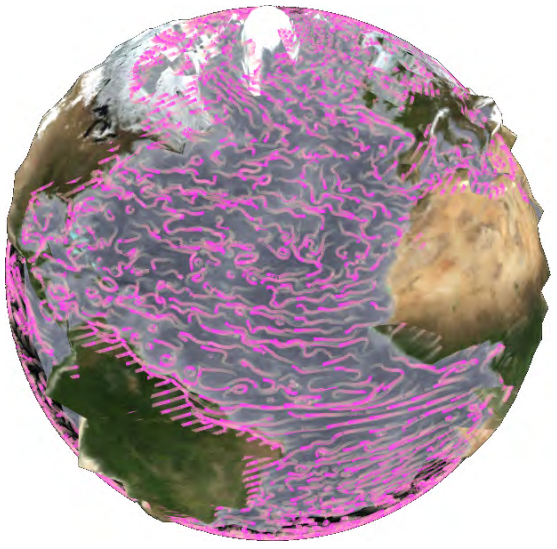


Figure 1: flow visualization of Atlantic ocean currents in WebGL. Hierarchical integration was used to reduce the total number of iterations required to calculate the paths.

With the growth of a great diversity of devices, development of multi-platform applications has become a common goal for developers. The Web is de-facto a universal platform to unify the development and execution of applications. However, challenges arise since applications must be optimized in order to be useful as well as on high as on low-performance devices.

The development of parallel computing hardware for all the different devices is increasing and the development of applications and computer-based procedures are taking in advance this capability. The contribution of this paper is a new methodology to calculate point trajectories of a highly dense grid of points over n-dimensional vector fields, in which the trajectories are forced to pass over the grid points (Figure 1). This allows to implement a hierarchical integration procedure ([HSW11]), which takes advance of previously calculated data in order to avoid repetitive and unnecessary calculations, and reduces the complexity of the algorithm from linear to logarithmic. The procedure is suitable to be implemented over highly parallel computing architectures due to independent calculations and the number of computations to be performed. We employ WebGL as the parallel computing engine to calculate the iterations, using the inherently parallel rendering procedure, and images are used to store the data through the iterations.

Different from other procedures, in which the calculation of the trajectories is performed for each point in particular, our methodology allows to merge its calculation for all the points in which the field is discretized. Therefore, the number of unnecessary computations is critically reduced.

This paper is organized as follows: Section 2 presents the related work. Section 3 exposes the methodology in which the contribution of this work is explained. Section 4 presents a case of study in oceanic currents and finally section 5 concludes the article.

2 LITERATURE REVIEW

2.1 Flow Visualization

A great amount of methodologies to visualize vector fields (flow fields) has been developed among the last decades. Geometric-based approaches draw icons on the screen whose characteristics represent the behavior of the flow (as velocity magnitude, vorticity, etc). Examples of these methodologies are arrow grids ([KH91]), streamlines ([KM92]) and streaklines ([Lan94]). However, as these are discrete approaches, the placement of each object is critical to detect the flow's anomalies (such as vortexes or eddies), and therefore, data preprocessing is needed to perform an illustrative flow visualization. An up-to-date survey on geometric-based approaches is presented by [MLP10].

However, in terms of calculating those trajectories for determined points in the field, the procedures usually compute for each point the integrals, and, as a result, the procedures are computationally expensive for highly dense data sets.

On the other hand, texture-based approaches represent both a more dense and a more accurate visualization, which can easily deal with the flow's feature representation as a dense and semi-continuous (instead of sparse and discrete) flow visualization is produced. A deep survey in the topic on texture-based flow visualization techniques is presented by [LHD04].

An animated flow visualization technique in which a noise image is bended out by the vector field, and then blended with a number of background images is presented by [VW02]. Then, in [VW03] the images are mapped to a curved surface, in which the transformed image visualizes the superficial flow.

Line Integral Convolution (LIC), presented by [CL93], is a widely implemented texture-based flow visualization procedure. It convolves the associated texture-pixels (texels) with some noise field (usually a white noise image) over the trajectory of each texel in some vector field. This methodology has been extended to represent animated ([FC95]), 3D ([LMI04]) and time varying ([LM05, LMI04]) flow fields.

An acceleration scheme for integration-based flow visualization techniques is presented by [HSW11]. The optimization relies on the fact that the integral curves (such as LIC) are hierarchically constructed using previously calculated data, and, therefore, avoid unnecessary calculations. As a result, the computational effort is reduced, compared to serial integration techniques, from $O(N)$ to $O(\log N)$, where N refers to the number of steps to calculate the integrals. Its implementation is performed on Compute Unified Device Architecture (CUDA), which allows a parallel computing scheme performed in the Graphics Processing Unit (GPU), and therefore the computation time is critically reduced. However, it requires, additionally to the graphic Application Programming Interface (API), the CUDA API in order to reuse data, and hence, execute the procedure.

2.2 WebGL literature review

The Khronos Group released the WebGL 1.0 Specification in 2011. It is a JavaScript binding of OpenGL ES 2.0 API and allows a direct access to GPU graphical parallel computation from a web-page. Calls to the API are relatively simple and its implementation does not require the installation of external plug-ins, allowing an easy deployment of multi-platform and multi-device applications. However, only images can be transfered between rendering procedures using framebuffer objects (FBOs).

Several WebGL implementations of different applications have been done such as volume rendering, presented by [CSK11] or visualization of biological data,

presented by [CADB10]. A methodology to implement LIC flow visualization with hierarchical integration, using only WebGL, was presented by [ACS12], in which FBOs are used to transfer data between different rendering procedures, and therefore allowing to take in advance the parallel computing capabilities of the rendering hardware, in order to perform the different calculations. However, for the best of our knowledge, no other implementation that regards to streamline flow visualization on WebGL has been found in the literature or in the Web.

2.3 Conclusion of the Literature Review

WebGL implementations allow to perform applications for heterogeneous architectures in a wide range of devices from low-capacity smart phones to high-performance workstations, without any external requirement of plug-ins or applets. As a result, optimized applications must be developed. In response to that, this work optimizes the calculation of point trajectories in n-dimensional vector fields over highly dense set of points, forcing the trajectories to lie over the points in the set. As a consequence, previously calculated data can be reused using hierarchical integration, avoiding unnecessary calculations and reducing the complexity of the algorithm.

3 METHODOLOGY

The problem that we address is stated as: given a set of points and a vector field that exists for all of these points, the goal is to find the finite trajectory that each point will reproduce for a certain period of time.

Normal calculation of point trajectories in n-dimensional vector fields, requires to perform numerical integration for each particular point in order to reproduce the paths. In the case of a dense set of points, the procedures suffer from unnecessary step calculations of the integrals, since several points in the field might lie over the same trajectory of others. Hence, some portions of the paths might be shared. Figure 2 illustrates this situation.

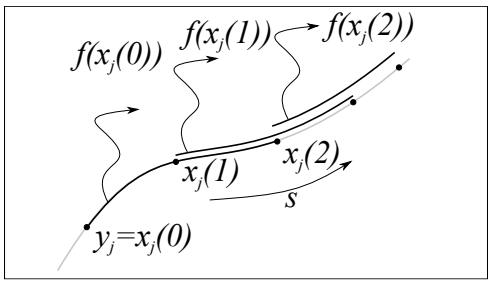


Figure 2: Trajectory overlapping in several point paths.

In order to avoid repeated computations, we propose a methodology to calculate trajectories of highly dense grid of points, in which the paths are forced to lie over the points in the grid, i.e., the paths are generated as a Piecewise Linear (PL) topological connection between a set of points that approximates the trajectory. With this, hierarchical integration [HSW11] is employed to iteratively compute the paths and reuse data through the iterations.

3.1 Hierarchical Integration

Since line integration over n-dimensional vector fields suffers from repeated calculations, hierarchical integration [HSW11] only calculates the necessary steps and then iteratively grows the integrals reusing the data. This reduces the computational complexity of the algorithm from $O(N)$, using serial integration, to $O(\log N)$. The procedure is summarized as follows.

For an arbitrary point in the field $y \in Y$ with $Y \subseteq \mathbb{R}^n$, let us define $f : Y \rightarrow \mathbb{R}^m$, as an arbitrary line integral bounded by its trajectory c_y. Consider its discrete approximation as described in equation 1.

$$f(y) = \int_{c_y} w(x(s))ds \approx \sum_{i=1}^{t} w(x(i*\Delta s))\Delta s \qquad (1)$$

where t is the maximum number of steps required to reproduce c_y with Δs the step size. $x(0) = y$ is the starting point of the trajectory to be evaluated and w is the function to be integrated. The integration procedure is performed for all points $y \in Y$ in parallel.

We assume that $\Delta s = 1$, $\forall y \in Y$ and therefore $f(y) \approx \sum_{i=1}^{t} w(x(i))$. The algorithm starts with the calculation of the first integration step for all the points in the field. Namely,

$$f_0(y) = w(x(1)) \qquad (2)$$

It is required to store the last evaluated point $x(1)$ over the growing trajectory and the partial value of the integral for all the points y in order to reuse them in the following steps to build the integral. With this, the next action is to update the value of the integral, using the sum of the previously calculated step at y and the step evaluated at its end point ($x(1)$). Namely,

$$f_1(y) = f_0(x(0)) + f_0(x(1)) \qquad (3)$$

In this case, the end point of $f_1(x(0))$ is $x(2)$ as the calculation evaluates $f_0(x(1))$. Therefore, the next iteration must evaluate f_1 at $x(0)$ and $x(2)$ in order to grow the integral. In general, the k'th iteration of the procedure is calculated as follows:

$$f_k(y) = f_{k-1}(x(0)) + f_{k-1}(x(end)) \qquad (4)$$

It is important to remark that each iteration of this procedure evaluates two times the integration steps evaluated in the previous iteration. As a result, the total number of integration steps t is a power of two,

and the hierarchical iterations required to achieve this evaluations is reduced by a logarithmic scale, i.e., $k = \log_2 t$. Also notice that the evaluation of the vector field is performed only once, in the calculation of the first step, which avoids unnecessary evaluations of the vector field, which are computationally demanding for complex vector fields. Figure 3 illustrates the procedure up to four hierarchical steps.

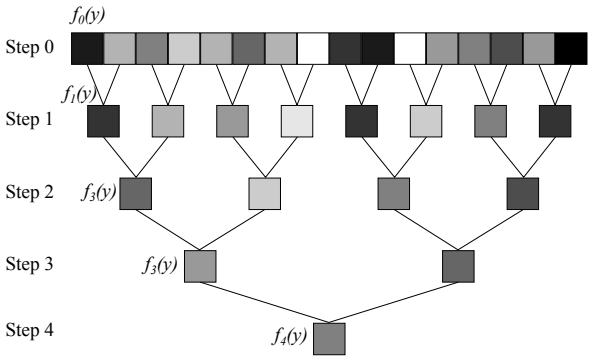


Figure 3: Exponential growth of hierarchical integration methodology. At step 3, the procedure evaluates 8 serial integration steps, meanwhile at step 4 it evaluates 16 serial integration steps.

3.2 Stream Path Calculation

In order to perform the visualization of a vector field using trajectory paths, lets assume a homogeneously distributed set of points

$$Y = \{y,\ z \in \mathbb{R}^n | y - z = \Delta y, \\ \Delta y \text{ is constant } \forall z \text{ adjacent to } y\} \quad (5)$$

and a n-dimensional vector field $F : \mathbb{R}^n \to \mathbb{R}^n$. The goal is to calculate for each point $y \in Y$, the PL approximation of the trajectory that the point will describe according to F, defined by the topological connection of a particular set of points $A_y \subset Y$. Figure 4 illustrates the approximation.

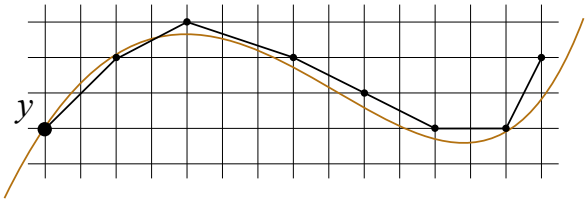


Figure 4: PL approximation of the trajectory by the procedure.

The trajectory c_y of an arbitrary point y in the field is defined as

$$c_y = x_y(s) = \int_l F(x_y(s))ds \quad (6)$$

where l represents a determined length of integration.

Using hierarchical integration, for each point in the field the first step of the PL trajectory is calculated, this is, the corresponding end point of the first step of the integral is computed using a local approximation of the point in the set of points.

$$x_y(0) = y \quad (7)$$

$$y' = x_y(0) + \gamma F(x_y(0)) \quad (8)$$

$$x_y(1) = \arg\min_{x_y}(Y - y') \quad (9)$$

where y' is the first iteration result of the Euler integration procedure, γ is a transformation parameter to adjust the step given by the vector field to the local separation of the set of points and $x_y(1)$ is defined as the closest point in Y that approximates y'. The value of $x_y(1)$ is then associated (and stored) to y. The set A_y contains the reference to the points of the trajectory that describes y, and therefore for equation 8, A_y is defined as:

$$A_y^0 = \{x_y(1)\} \quad (10)$$

Similarly to the hierarchical integration procedure, the next steps are performed to join the calculated steps in order to grow the trajectories. Therefore, for each point y, its computed trajectory is joint with its last point's trajectory, this is, for the step in equation 8.

$$A_y^1 = A_y^0 \cup A_{x_y(1)}^0 = \{x_y(1), x_y(2)\} \quad (11)$$

Note that each iteration of the procedure will increase the number of points in the trajectory by a power of two. Therefore, the growth of the paths is exponential. In general, the k'th iteration is calculated as

$$A_y^k = A_y^{k-1} \cup A_{x_y(2^k)}^{k-1} = \{x_y(1), x_y(2), \ldots, x_y(2^{(k+1)})\} \quad (12)$$

The accuracy of the procedure is strongly determined by the discretization (density) of Y, since it is directly related to the step size in the integration procedure, i.e., the approximation of the first step end-point is determinant. In order to increase the accuracy of the procedure, the computation of the first step can be calculated with e.g., a 4th order Runge Kutta numerical integration, however, it might significantly increase the computation time of the procedure if the computation time of the vector field function is relevantly high.

3.3 Time-Varying Data

In terms of unsteady flow fields, i.e., time-varying vector fields, the generation of the trajectories might seem difficult. In that case, as proposed in [HSW11], time is considered another dimension of the vector field.

Therefore, the set of points is formed with the position coordinates of the points and discretized time steps, producing an $n+1$-dimensional grid of points.

It is also determinant for the accuracy of the procedure that the density of the discretization set is high, in order to increase the precision of the approximated trajectories.

3.4 Animation

Dynamic scenes are demanding in most of the visualization procedures. We consider in this section two kinds of dynamic scenes. A first kind of procedures refers to when the vector field is steady, i.e., it remains constant through the time. In this case, the goal is to visualize the motion of the particle all across the field.

Since the paths for all the points in the field are calculated, the representation of the particle's trajectory through the frames is simple. Consider a point y and its approximated trajectory given by the set of points A_y. Notice, as described in section 3.2, that the first point of the set $A_y[1]$, i.e., $x_y(1)$, represents the next point in which y will lie in a determined period of time. As a result, at a posterior frame, the displayed trajectory should be $A_{x_y(1)}$.

The second type of procedure refers when vector field is varying with the time. Complementary to the animation stated before, a second kind of dynamic scene is comprised since it is also important to visualize the changes that a trajectory suffers in the time. In the case of time varying data, as in the steady case, all the points have an associated trajectory. In order to animate the change of one trajectory, from one frame to another, the trajectory that will be represented refers to the one of the point with the same point coordinate, but the next time coordinate. i.e., $A_{y,t+\Delta t}$.

4 CASE STUDY

In this section the visualization of 2D oceanic currents using the proposed methodology is performed. The implementation has been done in WebGL, so the methodology's parallel computing capabilities are fully used. WebGL offers the possibility to use the rendering procedure to calculate images (textures) through Framebuffer Objects, and then use those rendered textures as input images for other rendering procedures. As a consequence, for this implementation we associate the pixels of an image to the points in the field, and therefore, the rendering procedure is used to compute the different hierarchical iterations, which are stored in the color values of the pixels. Finally, the trajectories are hierarchically constructed. The implementation was performed on an Intel Core2Quad 2.33 GHz with 4 GB of RAM and with a nVidia GeForce 480.

4.1 Implementation

For a $w \times h$ grid of points (w and h being its width and height respectively in number of elements), images of size $w \times h$ in pixels are used, in which a particular pixel (i, j) is associated with the point (i, j) in the grid. Since for each particular pixel, a four component vector is associated, i.e., a vector of red, green, blue and alpha values, each value can be associated as a particular position of another pixel. This is, if the value of a pixel is r, then its associated pixel coordinates are given by

$$i = r \bmod w \tag{13}$$

$$j = \frac{r-i}{w} \tag{14}$$

where mod represents the remainder of the division of r by w. As a result, if for each hierarchical integration, only the last point of the trajectory is to be stored, then one image can store four hierarchical iterations.

For the zero'th hierarchical iteration, and the image I to store its calculation, the value of a pixel (i, j) is given by

$$i_0 = i + kF_i(i, j) \tag{15}$$

$$j_0 = j + kF_j(i, j) \tag{16}$$

$$\tag{17}$$

where the parameter '0' refers to the hierarchical step 0, k represents the scaling factor of the vector field, and $F_i(i, j)$ represents the component of the vector field over the direction of i, evaluated at the point (i, j). The vector field used in this case study is shown in figures 5(a) for the direction of i and 5(b) for the direction of j.

In general, the k'th step is calculated as follows,

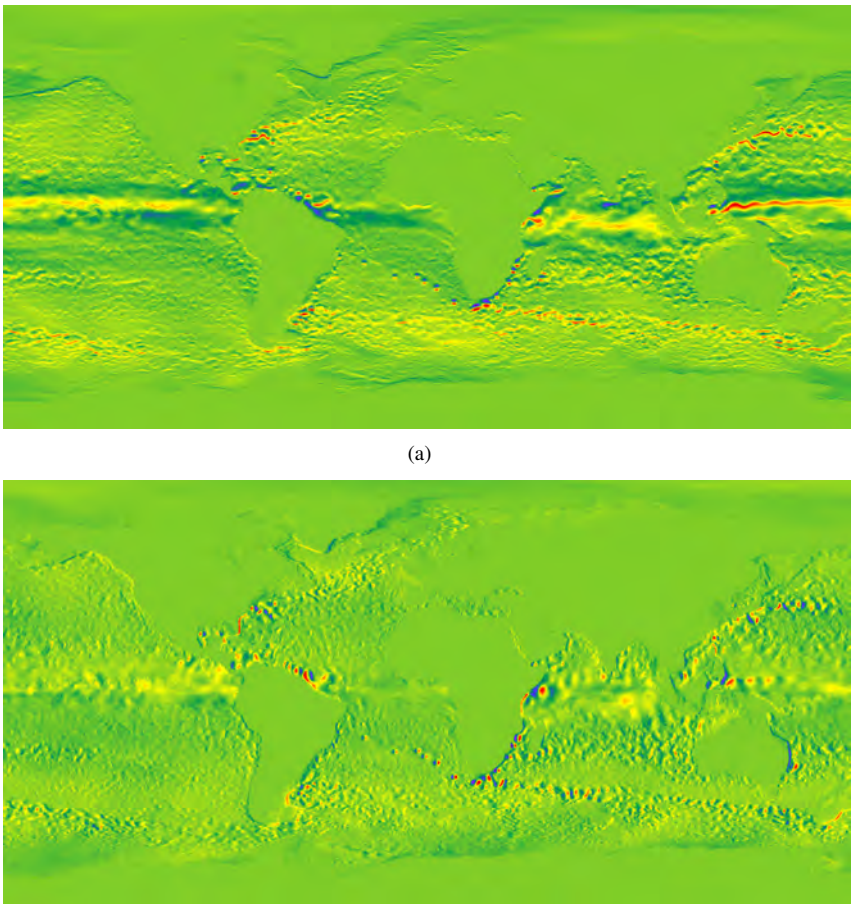
$$i_{next} = I(i, j, k-1) \bmod w \tag{18}$$

$$j_{next} = \frac{I(i, j, k-1) - i_{next}}{w} \tag{19}$$

$$I(i, j, k) = I(i_{next}, j_{next}, k-1) \tag{20}$$

In the case that k is greater than four, then more images are used to store the values of the hierarchical iterations.

With that, all the desired steps are stored in the necessary images. In order to build the trajectories from those images, a generic line, formed by k^2 points, is required. Each point in the trajectory needs to have an associated value, that refers to its order in the trajectory, i.e., the first point in the trajectory has an index 0, the second point the value 1 and so on. With this index associated to each point of the trajectory of the point y, the position of each point is calculated as described in Algorithm 1, where HL is the hierarchical level that needs to be evaluated and the function $evalHL()$ returns the new position of the point y, for a particular hierarchical level.



(a)

(b)

Figure 5: Oceanic currents information. Color-scale of the oceanic currents magnitude in the (a) longitudinal and (b) latitudinal directions, of the 1^{st} January of 2010. Images generated using data from the NASA's ECCO2 project.

Require: $index$ Index referring to the element position into the line.
y Position of the initial point of the trajectory.

Ensure: y_{end} Position of the i'th element of the line in the space.

Finished = $false$
while not Finished **do**
  $HL = \text{floor}(\log_2(index))$
  $index = index - 2^{HL}$
  $y = \text{evalHL}(HL, y)$
  if $index == 0$ **then**
    $y_{end} = y$
    Finished = $true$

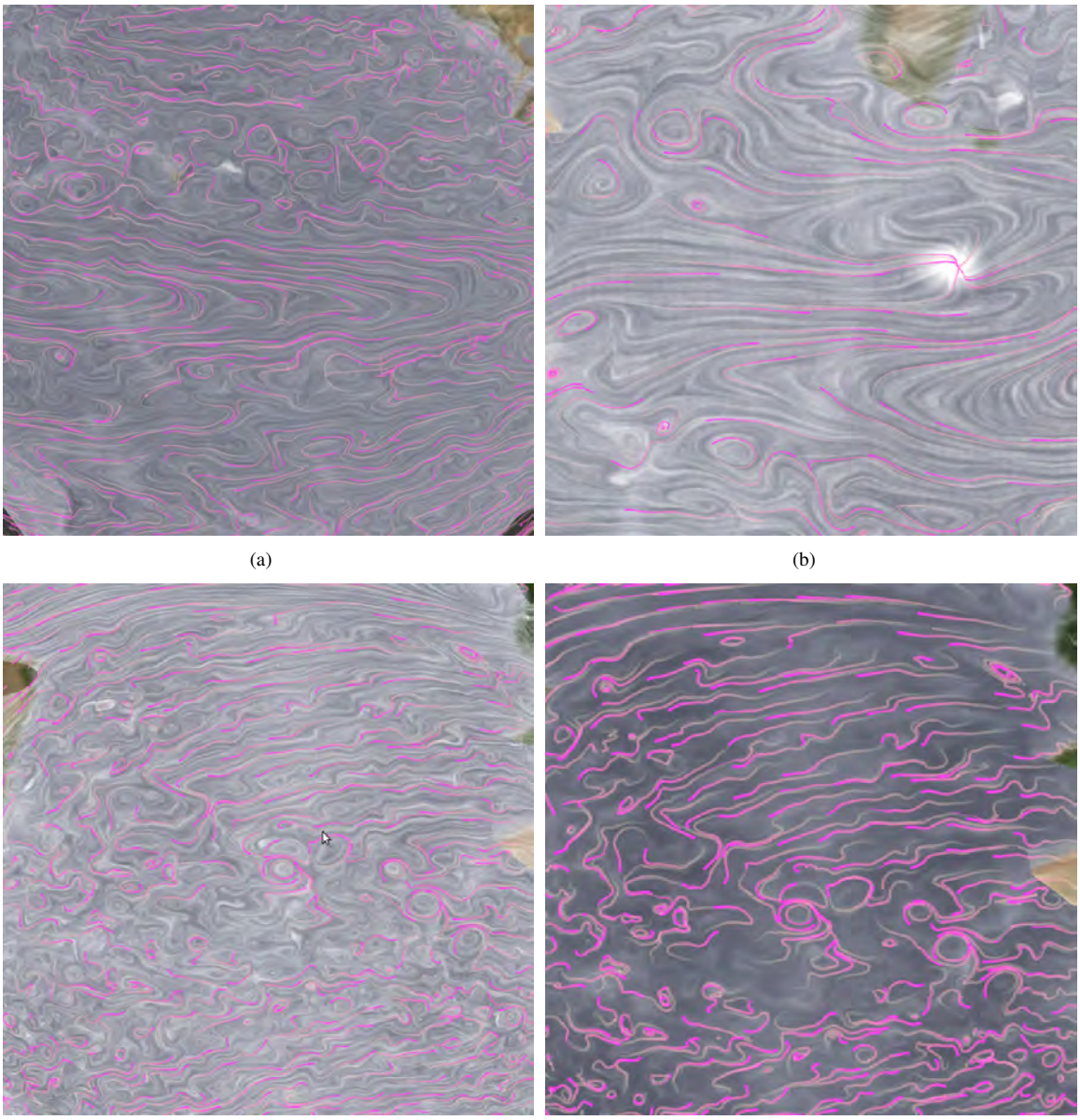
Algorithm 1: Procedure to reconstruct the streamlines using the already calculated hierarchical levels.

4.2 Results

A general grid of 2048×2048 points is used as the world's discretization. The vector field information was acquired from the NASA's ECCO2 project (see figure 5), in which high-resolution data is available. Only six hierarchical levels, i.e., $2^6 = 64$ points in the trajectory are used for each point in the field, as a result only 2 images are required to calculate the trajectories.

The time needed to compute all the hierarchical levels (from 0 to 6) was 3 ms. The trajectory computation was performed to 10000 equally-distributed points all over the field, which means that 64000 points need to be transformed by the trajectory computation. The computation time of those trajectories was 670 ms (Final results are shown in Figure 6).

In order to compare the visualization using the proposed methodology, the LIC visualization of the vector field using the methodology proposed in [ACS12] was inserted below the stream line visualization. It shows that for this level of discretization (2048×2048), the visualization is correct and accurate. However, sparse data might produce inaccurate results.



(a) (b)

(c) (d)

Figure 6: Different view points of the final visualization of the oceanic currents using hierarchically calculated streamlines. Six hierarchical steps were used to achieve this visualization. The LIC visualization of the flow is used below the streamlines in order to visually compare the different methods and to enhance the visualization

5 CONCLUSIONS AND FUTURE WORK

This article presents a novel methodology to calculate points trajectories in highly dense point sets, in which the trajectories are formed as piecewise-linear connections of the points in the set. This allows to merge the calculation of the different trajectories and use iteratively the data to avoid repeated and unnecessary calculations, hence, accelerating the process. The procedure is suitable to be implemented in parallel architectures such as WebGl, OpenCL or CUDA, since the calculations of the integrals for one point are independent from the calculation of the other points.

As a result, thanks to the use of hierarchical integration, the procedure reduces the computational complexity of the calculation of the trajectories from linear to logarithmic. The methodology deals with n-dimensional and time-varying data and animated visualization can be easily achieved due to the fact that the trajectories are calculated for all the points in the set.

Since the procedure performs an approximation of the trajectories using a piecewise-linear connection of the points in the set, the accuracy of the algorithm is strongly influenced by the discretization distance between the points, because this distance determines a lower bound in the integration step to be used.

Ongoing research focuses on the adaptation of this methodology to require less computational effort (processing and memory effort), so that extremely low-performance devices such as smart-phones and tablets might be able to perform an accurate and complete flow visualization using streamlines. Related future work includes the adjustment of the grid point positions along the iterations and the increase in the accuracy of the calculated trajectories.

ACKNOWLEDGEMENTS

This work was partially supported by the Basque Government's ETORTEK Project (ITSASEUSII) research program and CAD/CAM/CAE Laboratory at EAFIT University and the Colombian Council for Science and Technology COLCIENCIAS. The vector field information was acquired form NASA's ECCO2 project in *http://ecco2.jpl.nasa.gov/*.

6 REFERENCES

[ACS12] Mauricio Aristizabal, John Congote, Alvaro Segura, Aitor Moreno, Harbil Arriegui, and O. Ruiz. Hardware-accelerated web visualization of vector fields. case study in oceanic currents. In Robert S. Laramee Paul Richard, Martin Kraus and José Braz, editors, *IVAPP-2012. International Conference on Computer Vision Theory and Applications*, pages 759–763, Rome, Italy, February 2012. INSTICC, SciTePress.

[CADB10] M. Callieri, R.M. Andrei, M. Di Benedetto, M. Zoppè, and R. Scopigno. Visualization methods for molecular studies on the web platform. In *Proceedings of the 15th International Conference on Web 3D Technology*, pages 117–126. ACM, 2010.

[CL93] B. Cabral and L.C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270. ACM, 1993.

[CSK11] 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*, pages 137–146. ACM, 2011.

[FC95] L.K. Forssell and S.D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *Visualization and Computer Graphics, IEEE Transactions on*, 1(2):133–141, 1995.

[HSW11] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *Visualization and Computer Graphics, IEEE Transactions on*, (99):1–1, 2011.

[KH91] R.V. Klassen and S.J. Harrington. Shadowed hedgehogs: A technique for visualizing 2d slices of 3d vector fields. In *Proceedings of the 2nd conference on Visualization'91*, pages 148–153. IEEE Computer Society Press, 1991.

[KM92] D.N. Kenwright and G.D. Mallinson. A 3-d streamline tracking algorithm using dual stream functions. In *Proceedings of the 3rd conference on Visualization'92*, pages 62–68. IEEE Computer Society Press, 1992.

[Lan94] D.A. Lane. Ufat: a particle tracer for time-dependent flow fields. In *Proceedings of the conference on Visualization'94*, pages 257–264. IEEE Computer Society Press, 1994.

[LHD04] R.S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F.H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. In *Computer Graphics Forum*, volume 23, pages 203–221. Wiley Online Library, 2004.

[LM05] Z. Liu and R.J. Moorhead. Accelerated unsteady flow line integral convolution. *IEEE Transactions on Visualization and Computer Graphics*, pages 113–125, 2005.

[LMI04] Z. Liu and R.J. Moorhead II. Visualizing time-varying three-dimensional flow fields using accelerated uflic. In *The 11th International Symposium on Flow Visualization*, pages 9–12. Citeseer, 2004.

[MLP10] T. McLoughlin, R.S. Laramee, R. Peikert, F.H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum*, volume 29, pages 1807–1829. Wiley Online Library, 2010.

[VW02] J.J. Van Wijk. Image based flow visualization. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 745–754. ACM, 2002.

[VW03] J.J. Van Wijk. Image based flow visualization for curved surfaces. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 17. IEEE Computer Society, 2003.

Hybrid SURF-Golay Marker Detection Method for Augmented Reality Applications

David Prochazka, Ondrej Popelka, Tomas Koubek, Jaromir Landa, Jan Kolomaznik
Mendel University in Brno
Zemedelska 1
61300, Brno, Czech Republic
david.prochazka@mendelu.cz

ABSTRACT

Augmented reality is a visualization technique widely used in many applications including different design tools. These tools are frequently based on tracking artificial objects such as square markers. The markers allow users to add a 3D model into the scene and adjust its position and orientation. Nevertheless, there are significant problems with marker occlusions caused by users or objects within the scene. The occlusions usually cause a disappearance of the 3D model. Such behavior has substantial negative impact on the application usability. In this article we present a hybrid marker detection approach. With this approach, markers are detected using the well-known SURF method. This method is able to recognize complex natural objects and deal with partial occlusions. Further, we overcome the problem of distinguishing similar markers by using the Golay error correction code patterns. The described approach represents a robust method that is able to identify even significantly occluded markers, differentiate similar markers, and it works in a constant time regardless of the amount of used markers.

Keywords

Augmented reality, augmented prototyping, SURF, Golay error correction code.

1. INTRODUCTION

The augmented reality (AR) research has been running for almost two decades. Nevertheless, it is possible to find just a few applications for common users. There are several principal reasons. One of the key problems is the inability to deal with occlusions of markers that are used for scene augmentation. During the work with an AR application, a marker is frequently obstructed by different solid objects, e.g. **users´** hands. Inability to identify such **a** partially occluded marker leads to frequent disappearances of a visualized 3D model. Despite the obvious importance, this problem is unsolved even in many well-known AR toolkits (e.g. *ARToolKitPlus*).

The presented approach is implemented in the AR application *AuRel* that is focused on an augmented prototyping process**.** The application is developed in cooperation with an automotive company. It allows a car designer to extend a physical car model by selected virtual objects (3D models of car spare parts (see Fig. 1)). The usage of AR for industrial design is mentioned in many papers, e.g. in [FAM*02], [BKF*00] and [VSP*03].

Figure 1: 3D model of a spoiler inserted onto a rear car hood

Although there is a substantial amount of existing augmented reality frameworks (*ARToolkit*, *ARTag*, *Studierstube*, etc.), *OpenCV* library has been used for the implementation [Lag11]. Principal reasons being active *OpenCV* development, cross-platform

deployment, 64bit systems support, a wide range of implemented computer vision algorithms and the amount of documentation (books, tutorials, etc.) [PK11].

There are briefly summarized current methods used for recognition of possible markers in the section 2. Two approaches focused on identification of geometric features are compared with the advanced technique usually used for natural object detection. Further, the section 3 outlines our method that is composed of SURF marker detection and Golay error correction code identification. Finally, the section 4 presents our results and concentrates on the ability to deal with occlusions.

2. MARKER RECOGNITION METHODS

The process of marker recognition is usually divided in two parts: *marker detection* and *marker identification*. The former involves recognition of video frame regions that may represent markers. The latter concentrates on verifying the identity of the markers. The marker identity defines which 3D model will be displayed to the user.

2.1 Marker Detection Approaches

The registration process of all further described methods is influenced by many negative factors, e.g. low image resolution, camera distortion (caused by lens), various light conditions or marker occlusions. The methods endeavor to compensate most of these factors. For the purpose of the article, the methods are distinguished into three general groups according to their basic principles**.** In detail the description of object recognition methods can be found e.g. in [Sze11].

2.1.1 Morphology-based marker detection

These methods are based on recognition of shapes in preprocessed images. An approach described in [HNL96] uses a system of *concentric contrast circles* (CCC). The marker is composed of a black circle around a white middle or vice versa. The detection process starts with image thresholding and noise removal. Further, connected components are found, and their centers are determined. The results are two sets of centers: centers of white connected components and centers of black connected components. CCC marker position is given by the cross section of black and white centers**.**

An example of another approach is implemented in the frequently used *ARToolKit* [KB99]. In this case, square markers with black borders and black-and-white inner pictures are detected. A camera image is thresholded and connected components contours are found. Further, quadrangles are selected from the contours set. These quadrangles represent potential markers [KTB*03].

The obvious limitation of these methods is their inability to deal with occlusions. Such occlusion causes a change in the image morphology. Therefore, the required shape cannot be detected.

2.1.2 Edge-based marker detection

These methods are more flexible with regard to the marker occlusions than the image morphology-based methods. One solution that is based on this principle is the *ARTag* system [Fia05]. Although the *ARTag* markers are similar to the *ARToolKit* markers (significant is a thick black border), the implemented detection method is completely different. The *ARTag* method is based on detection of *edgels* (edge pixels) of an object. Further, a set of lines is constructed from the found *edgels*. It is not necessary to detect all line *edgels*; therefore, the edge could be partially occluded. Finally, four corresponding lines represent edges of a potential marker.

The same detection principle is used also in *StudierStube* project [Hir08] and many others.

2.1.3 Feature-based marker detection

These methods are based on key points detection. The key points are various regions of interest: edges, corners, blobs. To identify whether a given key point really represents a part of a marker, it is necessary to match it with a key point in a marker template. This matching process requires that both key points to be matched are described by gradient changes in their neighborhood. The process of key point

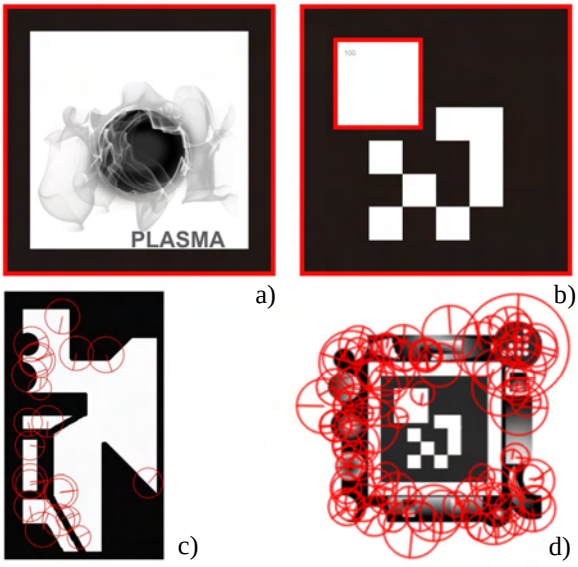


Figure 2: Markers used with different detection methods. Detected features are highlighted with a red color. From left: a) Match template, b) Golay error correction code, c) SURF and d) S-G detection

neighborhood description is usually called feature extraction. The output of this process is a set of feature descriptors vectors. The feature descriptors are later compared and their distance is computed [Low04]. This enables to match points of interest between a template and a camera image.

There are several approaches for feature-based detection. Widely used are e.g. SIFT [CHT*09] and SURF [BTG06]. A thorough comparison of selected methods is described in [TM08]. The SURF (Speeded Up Robust Features) algorithm has a good ratio between detection capabilities and performance. The SURF algorithm application is composed of three steps: detection of key points (points of interest), feature extraction and key point matching.

The detection of image key points that are used for the image description is based on gradient changes in the grayscale version of the image. Each key point is identified by position and radius that specifies the size of the key point neighborhood. Then the process of feature extraction is performed.

During this process, each key point neighborhood is described using 64-dimensional or 128-dimensional vector that describes the gradient changes of each key point neighborhood. The descriptors are produced both for a template and a camera image, so that the corresponding key points are identified.

The SURF main advantage is the scale and rotation invariance [BTG06]. This allows the SURF to work even with low resolution images or small objects. Another advantage is that the algorithm compares only the points; therefore, the object can be partially occluded. Although the SURF method is usually used for natural object identification (see e.g. [BCP*08]), it can be used also for marker detection as described in our method outlined in section 3.

2.2 Marker Identification Approaches

Morphological and edge-based detection methods are commonly used with following marker identification approaches: template matching and decoding of various binary codes.

Match template identification is based on computation of a pixel value correlation between a potential marker and a list of templates. In case the correlation fulfills a given threshold, the marker is identified. Obviously, the method has a linear time complexity. It is necessary to compute correlations with all templates until the required one is found or all templates are tested. Moreover, it is difficult to choose a threshold that allows to distinguish a large amount of markers [Bru09]. Therefore, methods based on different binary codes are frequently used to compensate this problem. One of the possible codes is the Golay error correction code.

A marker based on the Golay error correction code (ECC) can be composed of a large white square in the top left corner and e.g. 24 black or white squares that encode a number. The large square provides information about the marker orientation (see Fig. 2-b).

In the first step, a Golay ECC decoder for such a marker detects the position of the large white square. Further, it divides the code area into 24 blocks and calculates an average pixel value in all segments. Finally, the average value is thresholded to zero or one **a**nd the binary code is reconstructed. Possible implementation of the code reconstruction is outlined in [MZ06].

A significant advantage of this approach is that the binary code is reconstructed in a constant time. Another important advantage is the ability to correct errors caused by occlusions or an image corruption. Finally, the amount of distinguishable markers is limited just by the binary code length.

A feature-based method, such as the SURF is, is capable of both marker detection and marker identification. Therefore, it is not usually used with an identification method. As mentioned above, the method relies on searching for distinctive key points in a camera image that are then matched against image template descriptors. This process has linear time complexity because all template descriptors must be tested until the required one is found.

2.3 Summary of the Marker Recognition

In general, there are three approaches for marker recognition. The first one is based on image morphology. Detection can be fast and precise. However, it cannot deal with marker occlusions. The edge-based methods can detect partially occluded markers. Nevertheless, this ability is limited**.** Larger occlusions of the edges are problematic. Both detection methods can be accompanied by a binary code identification method that is able to work in a constant time and reliably distinguish a substantial amount of markers.

Feature-based approaches are able to detect and identify even substantially occluded markers. However, they work in a linear time. This complexity is usually problematic for real-time applications with larger amounts of markers. Even more, feature-based methods have problems with distinguishing of similar markers [SZG*09].

3. S-G HYBRID RECOGNITION METHOD

The proposed identification method combines the positive properties of two previously mentioned methods. We take advantage of robustness of the SURF feature-based object identification and

combine it with high reliability and effectiveness of the Golay error correction code detection, hence the name *S-G hybrid detection method*.

3.1 Marker design

As described in section 2.1.3, the SURF algorithm is suitable especially for natural objects identification. However, many applications use this method to identify only a single object in an image. This is marker may appear in a scene.

The most problematic part of the SURF marker identification is the matching of corresponding marker key points in both images. The key points similarity is determined by gradient changes in the key points neighborhoods (these are represented by feature descriptors). If the image contains areas with similar gradient changes, then such areas will be identified as the same or similar key points.

Therefore, it is important to design markers so that the key points identified in them have distinctive gradient changes. Furthermore, these key points must be distinguishable both from the scene image and from other markers.

We use artificial markers very distinctive from the scene objects. Acceptable results are obtained using complex asymmetric markers composed of arbitrary geometric shapes (see Fig. 2–c). These markers are easily detected because they contain a substantial amount of features which can be tracked. The development of such marker, however, requires a lot of manual work. Even with a thorough testing it seems that only a very low number (approx. 3) of these markers could be reliably distinguished in an image.

Therefore, to ensure the correct marker identification we propose a hybrid detection method – *S-G Detection* – in which we combine the SURF algorithm with the Golay error correction code. In this case, the marker template is divided into two parts: the marker border and marker content. These two parts of a template may be combined independently.

Marker content is composed solely of a Golay code image. Only the marker content is used for marker identification. This has the advantage of very high identification reliability and allows to distinguish large number of markers – see section 2.2.

Marker border is composed of different geometric shapes selected so that they are distinctive from real scene objects. However, the border is no longer used for identification of the markers. This is possible because each marker border may be combined with any number of Golay codes to identify the marker. This combination solves the problem of distinguishing between marker templates while maintaining great robustness against template occlusion (see Fig. 4). Different marker borders may be used in the application. However, it is not necessary. We use the same border for all markers.

3.2 Marker Detection

As been described in the previous section, we use the SURF method to identify key points only in the marker border (see Fig. 2–d). This border is the same for all markers. A strong advantage of this approach is that the time complexity of the whole algorithm is not a function of a number of templates (see section 2.1). Therefore, we can use a high number of markers without a performance hit. This is an important usability feature.

A common approach [Lag11] in matching the template and video frame points of interest is: find the best matches of key points using a defined metric, filter out false positives (invalid matches), repeat filtering until a sufficient number of adequately reliable points are obtained.

Errors in matched points may occur when a template key point is matched to an unrelated video frame key point because it happens to have similar neighborhood. Another source of errors occurs when a video frame contains two or more markers and template points are matched to correct points but on different marker borders (two or more physical markers).

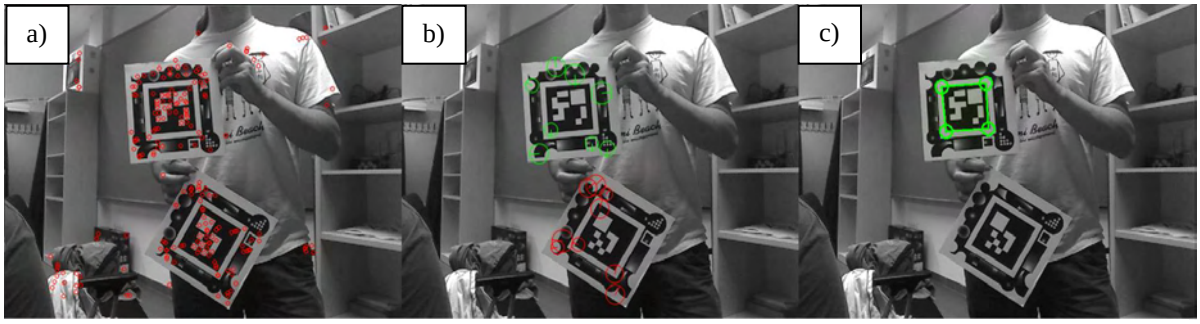


Figure 3: S-G hybrid method. From left: a) key points are detected and filtered b) angle filter is applied so that the key points on both markers are distinguished c) marker specified in application configuration is detected.

For many applications, it is enough to identify if the template is present in the image, other applications require approximate template positions. Our application requires the exact position (translation and rotation) of the marker so that the virtual object may be inserted to the real scene.

The first step of marker matching feature extractor is to discover key points in the processed image. Then a descriptor vector for each key point is found using a feature extractor. These vectors are matched by computation of Euclidean distance between each pair of points. Moreover, we use symmetric matching filter for the key points.

First, template key points are matched against video frame image and the best matches are selected. Then the frame key points are matched against template key points, and best matches are selected. The intersection of these two sets is a set of matched points [Lag11].

Further, we filter the set of key points by application of an angle filter. The idea behind the angle filter is to take advantage of the information stored in a SURF key point itself. Each SURF key point contains an angle value, which defines the direction of the most significant gradient descent in the neighborhood of the key point. In our application, we use artificial markers; therefore we search for a set of predefined objects. This means that relative differences in rotation of the matched key point must be similar for all matched key points. That is – if the template has two key points and their rotation is 45° and 70°, then the two key points matched in the frame must have the rotation difference approximately 25°. Due to perspective deformations, the differences can be computed only approximately. An example of this filtering is shown in Fig. 3 – each set of differently colored points maintains the same relative rotation differences between points (in other words the same rotation difference between a template and a video frame).

A difficult part of the angle filtering algorithm is defining initial conditions. This is caused by the fact that until the marker is identified, its key points, their rotations and order are all unknown. To overcome this problem, the angle filter algorithm is implemented by marshaling all possible rotations into overlapping intervals of a defined width (rotation difference tolerance – RT). Each interval overlaps half of neighboring intervals so that there are no artificial boundaries. Key points in each interval are then processed individually as if it was a standalone key point set. This introduces a performance hit as another loop iterating over sets of key point has to be processed. Fortunately this is upper bounded – maximum number of iterations is 360 / (RT · 2). This upper bound is hardly reached because only sets containing at least four points need to be processed. A minimum of four points is required for a correct positioning of a 3D model which will be added to the image later in the process. The angle filter algorithm may be described by the following pseudo-code:

```
FOR each matched_point
  difference =
    matched_point_template->angle -
    matched_point_frame->angle;
  div = difference / RT
  angles[div * RT]->add(matched_point)
  angles[(div + 1) * RT]
       ->add(matched_point)
END FOR
FOR each angle
  find homography
  identify Golay marker
  IF marker identified THEN
       display 3D object
END FOR
```

3.3 Marker Identification

For each set of points detected by the angle filter, we compute homography matrix so that the Golay code can be identified. By applying the homography transformation to the camera image we compensate the perspective deformation. This image transformed

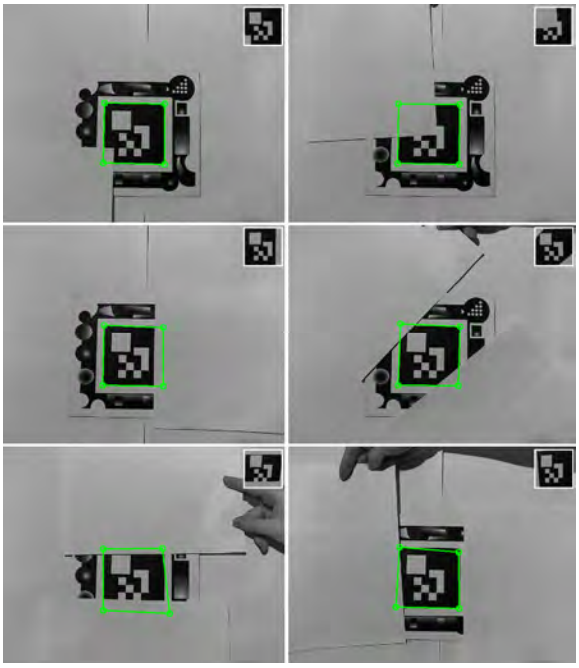


Figure4: Examples of S-G method capability of occluded marker identification from a close distance.

to the camera plane is cropped and processed by the Golay code detector.

If a Golay code is found, it means that the marker is identified. This identification introduces important feedback for the SURF marker detection. Given the reliability of the Golay detector, false positives are almost impossible. In other words, if the code is identified, we can be sure it is one of searched markers. It also means that the homography was computed correctly. This is also important because we can use the points to compute projection matrix. Reliable projection matrix is important for correct 3D models positioning.

In section 2.2 that describes the Golay codes is stated that the Golay code rotation is determined by the position of the large white square in the top left corner. Since the S-G detection method is focused on robustness against marker occlusions, it is undesirable to have parts of the marker with greater importance. In the S-G method, the rotation of the marker is determined solely by the position of key points. This part of the Golay code is therefore unused.

4. COMPARISON

The S-G hybrid method was tested against two other solutions: *ARToolKitPlus* (http://handheldar.icg.tugraz.at/artoolkitplus.php) and *ALVAR Toolkit* (www.vtt.fi/multimedia/alvar.html). All tests were made in a laboratory under artificial light. We used markers with 14 cm long edge for testing. The solutions were tested from three aspects:

- Distance – minimum, maximum and maximum distance without visible jitter.
- Angles – marker was placed at different distances from the camera and rotated around *x* and *y* axis (the *z* axis was not tested because all solutions are capable of 360 degrees rotation).
- Occlusion – occlusion was tested with stationary marker and camera.

Compared to the other two solutions, S-G has a smaller maximum distance where it is capable to identify a marker. The S-G method is able to detect a marker placed at a distance 2 m from the camera. The *ARToolKitPlus* and *ALVAR* have maximal distance at approx. 5 m.

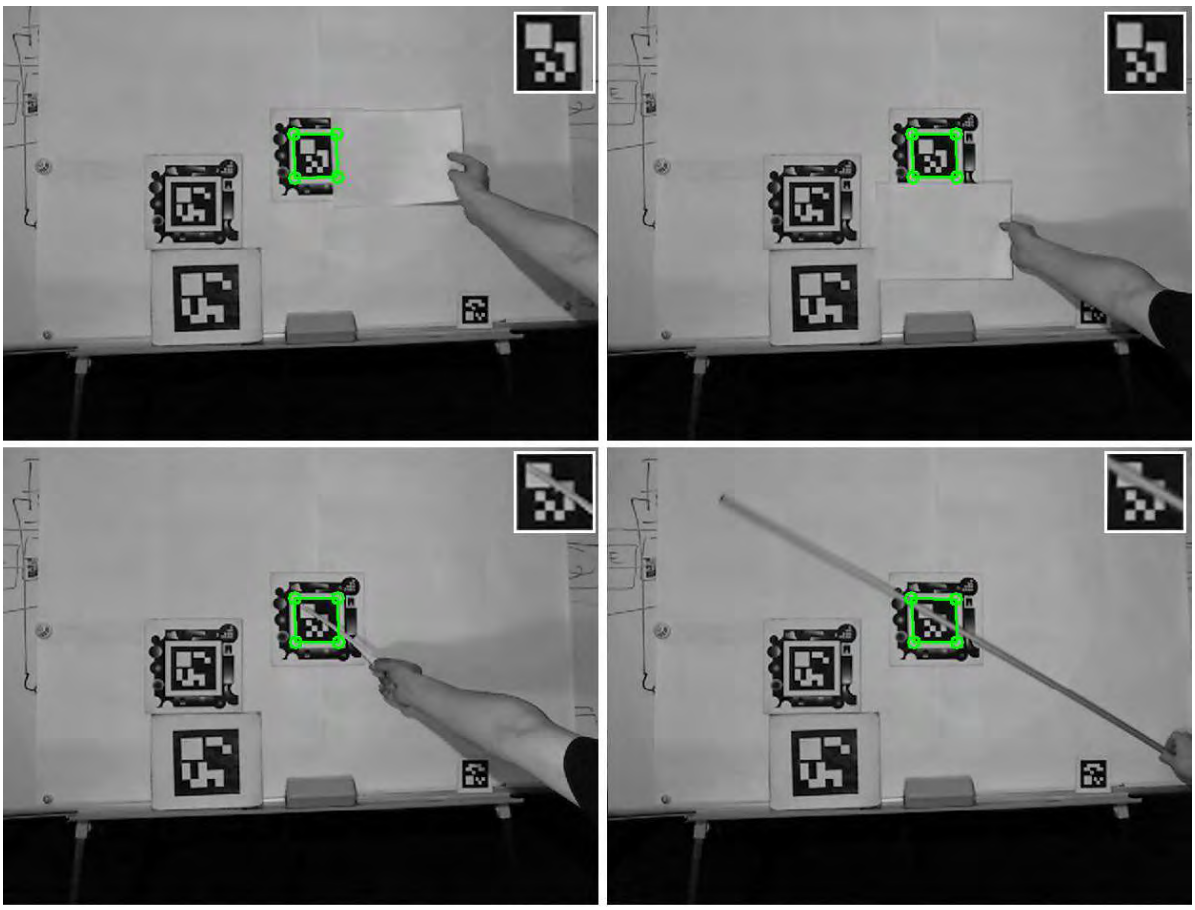


Figure 5: Examples of S-G method capability of occluded marker identification from a large distance. Both the marker boarder and marker content may be occluded.

In the angles comparison, measured results are influenced by the SURF algorithm limitations. The S-G method is able to detect a marker that is under 55° angle to the camera axis. (0° represents a marker perpendicular to the camera axis. The maximal theoretical angle is therefore 90°.) The other two solutions have maximal angles ranging from 74° to 85°.

Neither *ARToolKitPlus* nor *ALVAR* can deal with any type of occlusion. This is the most important disadvantage of these solutions. The S-G method can deal with significant marker occlusion. Because S-G works with key points instead of morphological operations or e.g. *edgels*, it is able to withstand a substantial number of different occlusions. We tested several of them (see Fig. 4).

The marker border can be obstructed up to 50 %. It is irrelevant what part of marker border is obstructed (all corners, two whole sides, etc.). The marker content (the Golay error correction code) must be visible at least from 75 % in case the large white square is obstructed. In case the obstruction is in other part of the Golay code, maximum allowed occlusion is approx. 15 %. This occlusion is limited by the Golay code redundancy.

This is the most important contribution of our solution in comparison to other used methods**.**

Because of the nature of the detection, the solutions capable of occlusion (e.g. *ARTag*) need at least three visible marker corners to detect and identify the marker. Our method is capable of the identification of a marker with all corners or sides covered. Our method has capability even of overcoming of the occlusion of marker contents. This is possible because of the Golay error correction code usage.

Figure 6: Marker occlusion. The marker is approx. 2 m distant from the camera.

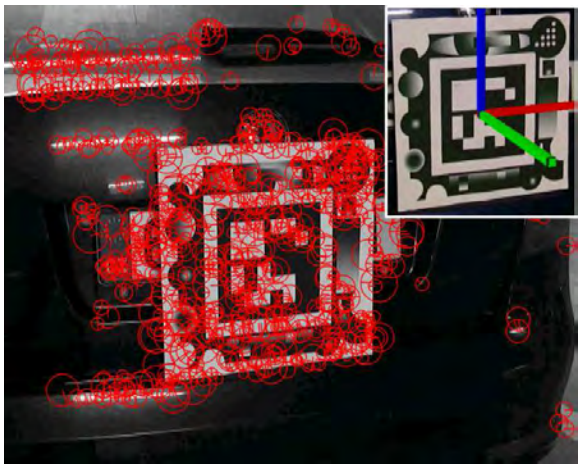


Figure 7: Key points detected by the S-G method and augmented 3D model.

5. CONCLUSION

Our application aims to improve the car design process. Therefore, several criteria must be fulfilled: Our marker detection and identification methods must be able to distinguish several hundred markers (one marker represents one spare part). Further, it must be possible to compute the precise position and rotation of the marker. Finally, the methods must be able to deal with occlusions that are common in real situations.

The SURF detection method as well as the Golay error correction code is able to deal with the occlusions. The proposed S-G registration method is slower than other frequently used approaches (e.g. image morphology approach with the Golay error correction codes). Still, it works in a constant time that is significant for real-time applications.

Nevertheless, in case of very good lighting conditions and absence of occlusions we recommend techniques based on the image morphology. With these methods, the video stream processing speed is substantially improved. Our *AuRel* application supports both approaches; therefore, registration technique is chosen according to the current conditions. By default, the morphology-based method (16 fps) is used. In case a marker detected in previous frame is missing, we switch to the S-G method (4 fps). Following frame is again processed by morphology-based method. Frame rates are measured on 640×480 px camera stream processed by Intel Core i5 2.6 GHz, 4 GB RAM, HDD 7200 rpm.

We consider our approach very promising. Nonetheless, there must be further research focused on several technical aspects. Particularly, the marker detector performance should be optimized (on the reference hardware configuration, *ARToolKitPlus* and *ALVAR* have above 20 fps). This could be done by reducing the number of key points in exchange for

lower reliability. Also the maximum detection distance needs to be improved. Possible solution can be to improve marker design so that the marker detector response is increased as outlined in [Sch*09].

SURF method can be easily used to design a marker-less tracking method as outlined in many articles. The absence of markers can substantially improve the application usability. Nevertheless, there could be a problem with selection of a correct 3D model and its manual position adjustment.

6. ACKNOWLEDGMENTS

This paper is written as a part of a solution of the project IGA FBE MENDELU 7/2012 and FBE MENDELU research plan MSM 6215648904.

7. REFERENCES

[BCP*08] Barandiaran, I., Cottez, Ch., Paloc, C., Grana, M.: Comparative Evaluation of Random 159 Forest and Ferns Classifiers for Real-Time Feature Matching in WSCG 2008 Full Papers Proceedings, Plzen: University of West Bohemia, 2008, pp. 159-166.

[Bru09] Brunelli, R.: Template Matching Techniques in Computer Vision: Theory and Practice, Wiley, 2009, ISBN 978-0-470-51706-2.

[BKF*00] Balcisoy, S., Kallmann, M., Fua, P., Thalmann, D.: A framework for rapid evaluation of prototypes with Augmented Reality. In Proceedings of the ACM symposium on Virtual reality software and technology, pp. 61-66. 2000. ISBN:1-58113-316-2.

[BTG06] Bay, H., Tuytelaars, T., Gool, L. V.: Surf: Speeded up robust features. In ECCV, 2006, pp. 404–417.

[CHT*09] Cui, Y., Hasler, N., Thormahlen, T., Seidel, H.: Scale Invariant Feature Transform with Irregular Orientation Histogram Binning. In Proceedings of Image Analysis and Recognition: 6th International Conference, pp. 258-267, 2009. ISBN: 978-3-642-02610-2.

[FAM*02] Fiorentino, M., De Amicis, R., Monno, G., Stork, A.: Spacedesign: A Mixed Reality Workspace for Aesthetic Industrial Design. In Proceedings of International Symposium on Mixed and Augmented Reality, p. 86. 2002. ISBN:0-7695-1781-1.

[Fia05] Fiala, M.: ARTag, a fiducial marker system using digital techniques. Computer Vision and Pattern Recognition, 2, June, 2005.

[Hir08] Hirzer, M.: Marker detection for augmented reality applications, 2008. Inst. for Computer Graphics and Vision, Graz University of Technology, Austria.

[HNL96] Hoff, W. A., Nguyen, K., Lyon T.: Computer vision-based registration techniques for augmented reality. In Intelligent Robots and Computer Vision, XV, 1996, pp. 538–548.

[KB99] Kato, H., Billinghurst, M.: Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality. 1999, s. 85-94.

[KTB*03] Kato, H., Tachibana, K., Billinghurst, M., Grafe, M.: A registration method based on texture tracking using artoolkit. In Augmented Reality Toolkit Workshop, 2003. IEEE International, 2003, IEEE, pp. 77–85.

[Lag11] Laganiere, R.: OpenCV 2 Computer Vision Application Programming Cookbook. Packt Publishing, 2011.

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

[MZ06] Morelos-Zaragoza, R. H.: The Art of Error Correcting Coding (Second Edition). John Wiley & Sons, 2006.

[PK11] Prochazka, D., Koubek, T.: Augmented Reality Implementation Methods in Mainstream Applications. Acta of Mendel University of agriculture and forestry Brno 59, 4,. 2011, p. 257.

[Sch*09] Schweiger, F. et al.: Maximum Detector Response Markers for SIFT and SURF, Proceedings of the Vision, Modeling, and Visualization Workshop 2009, Germany.

[SZG*09] Schweiger, F., Zeisl, B., Georgel, P., Schroth, G., Steinbach, E., Navab, N.: Maximum Detector Response Markers for SIFT and SURF. In Int. Workshop on Vision, Modeling and Visualization (VMV), 2009.

[Sze11] Szeliski, R.: Computer Vision: Algorithms and Applications, Springer, 2011.

[TM08] Tuytelaars T., Mikolajczyk K.: Local invariant feature detectors: a survey, Foundations and Trends® in Computer Graphics and Vision archive, Volume 3 Issue 3, 2008, pp. 177-280.

[VSP*03] Verlinden, J. C., Smit, A. D., Peeters, A. W. J., Gelderen, M. H. V.: Development of a flexible augmented prototyping system. Journal of WSCG, 11, 2003, pp. 496-503

A Survey of Cloud Lighting and Rendering Techniques

Roland Hufnagel
Univ. Salzburg, Salzburg, Austria
rhufna@cosy.sbg.ac.at

Martin Held
Univ. Salzburg, Salzburg, Austria
held@cosy.sbg.ac.at

ABSTRACT

The rendering of participating media still forms a big challenge for computer graphics. This remark is particularly true for real-world clouds with their inhomogeneous density distributions, large range of spatial scales and different forms of appearance. We survey techniques for cloud visualization and classify them relative to the type of volume representation, lighting and rendering technique used. We also discuss global illumination techniques applicable to the generation of the optical effects observed in real-world cloud scenes.

Keywords

cloud rendering, cloud lighting, participating media, global illumination, real-world clouds

1 INTRODUCTION

We review recent developments in the rendering of participating media for cloud visualization. An excellent survey on participating media rendering was presented by Cerezo et al. [5] a few years ago. We build upon their survey and present only recently published techniques, with a focus on the rendering of real-world cloud scenes. In addition, we discuss global illumination techniques for modeling cloud-to-cloud shadows or inter-reflection.

We start with explaining real-world cloud phenomena, state the graphics challenges caused by them, and move on to optical models for participating media. In the following sections we categorize the state-of-the-art according to three aspects: The representation of clouds (Section 2), rendering techniques (Section 3) and lighting techniques (Section 4).

1.1 Cloud Phenomenology

Clouds exhibit a huge variety of types, differing according to the following aspects.

Size: Clouds reside in the troposphere, which is the layer above the Earth's surface reaching up to heights of 9–22 km. In the mid-latitudes clouds show a maximum vertical extension of 12–15 km. Their horizontal extension reaches from a few hundred meters (Fig. 1.7) to connected cloud systems spanning thousands of kilometers (Figs. 1.11 and 1.12).

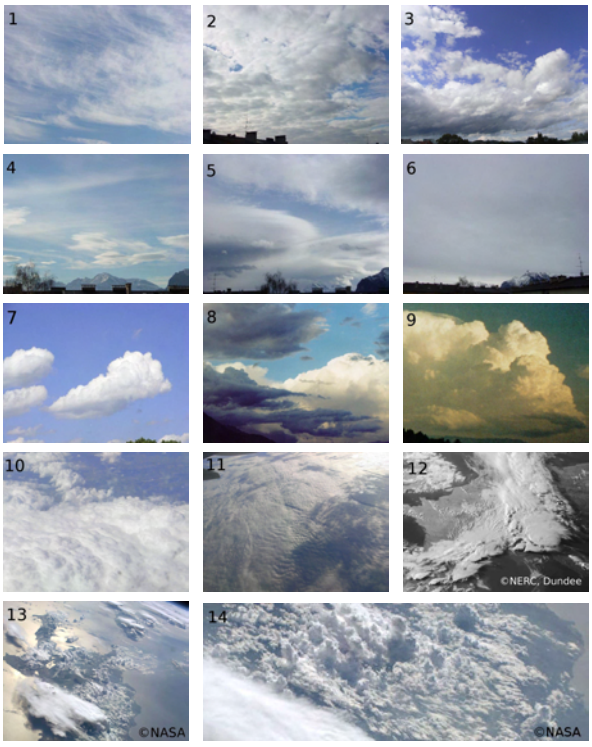


Figure 1: Clouds as seen from the ground: broken cloud layers with fractal cloud patches (top row), clouds with diffuse boundaries (second row), dense cloud volumes with surface-like boundaries (third row); and clouds viewed from a plane and from space (last two rows), where (14) shows a zoom into the central region of (13).

Geometry: In relation to the Earth's radius (6371 km) the troposphere represents a shallow spherical shell. The curvature of the Earth produces the horizon and is directly visible when viewing clouds from space.

Clouds often develop at certain heights and form layers. These either consist of cloud patches (Fig. 1.1 to 1.5) or create overcast cloud sheets (Fig. 1.6 or 1.11).

Local upward motions emerging from the ground (convective plumes) create clouds with a sharp cloud base and a cauliflower-like structure above, so-called Cumulus (Figs. 1.7 to 1.9). Their vertical extension reaches up to several kilometers (Fig. 1.9).

For low clouds also mixed forms of convective and layered clouds exist, where the convective plumes are vertically bound and form a layer (Fig. 1.3).

Clouds formed in connection with frontal systems usually show a large vertical extension and can span thousands of kilometers (Fig. 1.12).

Boundary Appearance: The appearance of clouds mainly depends on the cloud volume's constituents: droplets of different size or ice particles. While large water droplets produce a determined, surface-like boundary (Figs. 1.7 to 1.9), smaller droplets create a diffuse or fractal boundary (Figs. 1.2 to 1.6). Ice particles often form hair-like or fiber-like clouds, so-called Cirrus (Fig. 1.1), or diffuse clouds, the anvil-like tops of convective clouds (Fig. 1.13).

The appearance of a cloud is strongly influenced by the distance to its observer: While distant clouds often show a distinct surface and sharp contours (Figs. 1.12 and 1.13), a closer look reveals diffuse or fractal structures (Fig. 1.14).

Optical Phenomena: Clouds consist of water droplets and ice crystals which scatter light mostly independent of wavelength. Clouds are therefore basically white. Spectral colors appear only at certain angular constellations but generally do not influence their overall appearance. However, several optical phenomena determine their characteristic, natural appearance:

Self-Shadowing: The attenuation of light within a cloud creates gray tones and is proportional to the optical depth of the volume. The self-shadowing provides the cue to perceive clouds as volumetric objects.

Multiple scattering slightly attenuates the effect of self-shadowing by distributing light within the cloud volume in a diffusion-like process; see e.g. [4, 27, 33].

Inter-Cloud Shadows: Clouds cast shadows onto other clouds, like in Fig. 1.11, where a high cloud layer on the right-hand side shadows a low cloud layer.

The Earth's Shadow: Clouds can be shadowed by the Earth; see Fig. 1.8 showing an evening scene, where the low clouds lie in the shadow of a mountain range.

Indirect Illumination: Light inter-reflection between different clouds or between different parts of the same cloud brighten those regions, as, e.g., in Fig. 1.9, where the cloud seems to gloom from the inside.

Light Traps: Light inter-reflection at a smaller scale occurs on determined cloud surfaces (Neyret [29]) and lets concavities appear brighter (Figs. 1.7 and 1.10).

Corona: The corona effect occurs when the cloud is lit from behind. The strong forward scattering at the boundary produces a bright silhouette (silver-lining).

Atmospheric Scattering: Clouds often appear in vivid colors. This is caused by the scattering of light outside the cloud volume on air molecules and aerosols. Clouds are therefore often lit by yellowish to reddish sunlight (Fig. 1.9). Different paths of light in the atmosphere let the high clouds in Fig. 1.14 appear bright white and the low clouds yellowish. Atmospheric scattering also creates blue skylight which in some situations represents the main source of lighting (Fig. 1.8).

Ground Inter-Reflection: For low-level clouds the inter-reflection with the ground creates subtle tones depending on the type of the ground; see e.g. [3].

1.1.1 Summary

From a computer graphics point of view we identify the following volume properties, in addition to the form of the clouds as created by cloud modeling techniques, which are out of the scope of this survey: **thin** clouds that show no self-shadowing; cloud patches that represent a mixture of slightly dense cores and optically thin boundaries, usually forming horizontally extensive **layered** clouds; and **dense** cloud volumes of different sizes and extensions with a sharp or surface-like boundary. The boundary of clouds exhibits either a **fractal**, **diffuse** or **sharp** appearance.

1.2 Computer Graphics Challenges

A realistic visualization of clouds requires to tackle the following challenges:

Heterogeneity: Real-world cloud scenes typically consist of a very heterogeneous collection of clouds with different appearances, sizes and forms. Different cloud types require different volume representations, lighting and rendering techniques.

Atmospheric Scattering: For creating a cloud's natural appearance the lighting model employed has to reproduce its typical optical phenomena (see Sec. 1.1), including atmospheric scattering which is the main source for color in the sky. This requires the inclusion of atmospheric models (which are not discussed in this survey) in the lighting process of clouds.

Curved volume: The spherical atmosphere makes it difficult to take advantage of axis-aligned volumes if clouds are viewed on a large or even global scale, as in Figs. 1.12 or 1.13.

Huge domain: The sheer size of the volume of real-world cloud scene, especially when viewed from above, like in Figs 1.11 to 1.13, requires sophisticated and efficient lighting and rendering techniques.

1.3 Participating Media

A cloud volume constitutes a participating medium exhibiting light attenuation and scattering. This section uses the terminology of [5] to provide a short introduction to the radiometry of participating media.

While light in vacuum travels along straight lines, this is not the case for participating media. Here photons interact with the medium by being scattered or absorbed. From a macroscopic point of view light spreads in participating media, gets blurred and attenuated, similar to heat diffusion in matter.

Participating media are characterized by a particle density ρ, an absorption coefficient κ_a, a scattering coefficient κ_s, and a phase function $p(\vec{\omega},\vec{\omega}')$ which describes the distribution of light after scattering.

Absorption is the process where radiation is transformed to heat. The attenuation of a ray of light with radiance L and direction $\vec{\omega}$ at position x (within an infinitesimal ray segment) is described by

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = -\kappa_a(x)\,L(x,\vec{\omega}).$$

In the atmosphere absorption is mainly due to water vapor and aerosols. Cloud droplets or ice crystals show little absorption which means that the light distribution in clouds is dominated by scattering.

Scattering is the process where radiance is absorbed and re-emitted into other directions. *Out-scattering* refers to the attenuation of radiance along direction $\vec{\omega}$ due to scattering into other directions:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = -\kappa_s(x)\,L(x,\vec{\omega}).$$

In-scattering refers to the scattering of light into the direction $\vec{\omega}$ from all directions (integrated over the sphere) at a point x:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = \frac{\kappa_s(x)}{4\pi}\int_{4\pi} p(\vec{\omega}',\vec{\omega})\,L(x,\vec{\omega}')\,d\omega',$$

Extinction is the net effect of light attenuation due to absorption and out-scattering described by the extinction coefficient $\kappa_t = \kappa_a + \kappa_s$.

Emission contributes light to a ray:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = \kappa_a(x)\,L_e(x,\vec{\omega}).$$

It is usually not relevant for cloud rendering since clouds do not emit light. (An exception is lightning inside a cloud.)

Radiative Transfer Equation (RTE): Putting all terms together yields the RTE which describes the change of radiance within a participating medium at a point x:

$$\begin{aligned}(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) \;=\;& \kappa_a(x)\,L_e(x,\vec{\omega}) + \\ & \frac{\kappa_s(x)}{4\pi}\int_{4\pi} p(\vec{\omega}',\vec{\omega})\,L(x,\vec{\omega}')\,d\omega' - \\ & \kappa_a(x)\,L(x,\vec{\omega}) - \kappa_s(x)\,L(x,\vec{\omega}).\end{aligned}$$

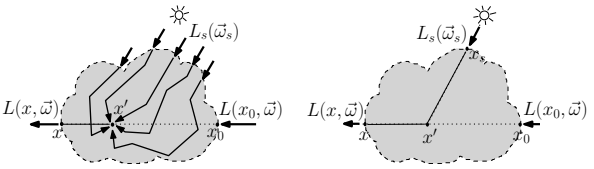


Figure 2: Multiple scattering (left), and the single scattering approximation (right).

By using the single scattering albedo $\Omega = \kappa_s/\kappa_t$ and noting that κ_a can be expressed as $\kappa_a = \kappa_t(1-\Omega)$, we can re-write the RTE as

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = \kappa_t(x)\,J(x,\vec{\omega}) - \kappa_t(x)\,L(x,\vec{\omega}), \quad (1)$$

with the source radiance J:

$$\begin{aligned}J(x,\vec{\omega}) \;=\;& (1-\Omega(x))\,L_e + \\ & \frac{\Omega(x)}{4\pi}\int_{4\pi} p(\vec{\omega}',\vec{\omega})\,L(x,\vec{\omega}')\,d\omega'.\end{aligned}$$

The source radiance describes all contributions of radiance to a ray $(x,\vec{\omega})$ at a point x inside the medium. In high-albedo media, like clouds, the source term is mainly due to in-scattering, while the extinction is dominated by out-scattering: $\kappa_t \approx \kappa_s$.

Integrating the RTE along a ray from x_0 to x yields the radiance reaching point x from direction $-\vec{\omega}$ (see Fig. 2, left):

$$L(x,\vec{\omega}) = T(x,x_0)L(x_0,\vec{\omega}) + \int_{x_0}^{x} T(x,x')\,\kappa_t(x')\,J(x')\,dx', \quad (2)$$

with the transmittance $T(x_1,x_2) = \exp(-\int_{x_1}^{x_2}\kappa_t(x)\,dx)$. The boundary condition of the integral is $L(x_0,\vec{\omega})$, representing the light coming from the background, an environment map, or from scene objects.

Note that the coefficients κ_* depend on the wavelength. Therefore, three versions of Eqn. 2 have to be solved with appropriate coefficients $\kappa_{*,\lambda}$ for each wavelength corresponding to the RGB color components.

Single Scattering Approximation: A difficulty in solving Eqn. 2 is that L appears (implicitly through J) on both sides of the equation. A common approximative solution is to account only for a certain number of scattering events and apply extinction on the paths in between. Considering only the first order of scattering yields the single scattering approximation: The source radiance J_{SS} is given by the light from the light source L_s attenuated on its way between x_s and x', see Fig. 2 (right), thus eliminating L:

$$J_{SS}(x',\vec{\omega}) = \Omega(x')\,T(x',x_s)\,p(x',\vec{\omega}_s,\vec{\omega})\,L_s(x_s,\vec{\omega}_s).$$

The single scattering approximation simulates the self-shadowing of a volume. Higher order scattering accounts for the "more diffuse" distribution of light within the volume.

Phase Function: A scattering phase function is a probabilistic description of the directional distribution of scattered light. Generally it depends on the wavelength, and the form and size of the particles in a medium. Usually phase functions show a symmetry according to the incident light direction, which reduces it to a function of the angle θ between incident and exitant light $p(\theta)$.

Often approximations for certain types of scattering are used, like the *Henyey-Greenstein* or the *Schlick* function. However, as noted in [4], those functions cannot model visual effects that depend on small angular variations, like glories or fog-bows. See [4] and its references for plots and tabular listings of phase functions.

2 CLOUD REPRESENTATIONS

A cloud representation specifies the spatial distribution, overall structure, form, and boundary appearance of clouds in a cloud scene.

2.1 Hierarchical Space Subdivision

2.1.1 Voxel Octrees

Voxel octrees are a hierarchical data structure built upon a regular grid by collapsing the grid cells of a $2 \times 2 \times 2$ cube (children) to a single voxel (parent).

Sparse voxel octrees reduce the tree size by accounting for visibility and LOD: Interior voxels are removed, yielding a hull- or shell-like volume representation (see Fig. 3, middle). Also hidden voxels (relative to the current viewing point) are removed (see Fig. 3, right). Additionally the LOD resolution can be limited according to the screen resolution (view-dependent LOD). These techniques require an adaptive octree representation accompanied with an update strategy.

Crassin et al. [7], and similarly Gobetti et al. [13], propose a dynamic octree data structure in combination with a local regular grid representation. Each node of the octree is associated with a *brick*, a regular 32^3 voxel grid, which represents a filtered version of the volume enclosed by its child nodes. The bricks are stored in a *brick pool* of a fixed size. Bricks are referenced by pointers stored in the nodes of the octree (see Fig. 4). The octree nodes themselves are stored in a pool as well (*node pool*). During the visualization brick and node data is loaded on demand and the pools are managed by updating least recently used data.

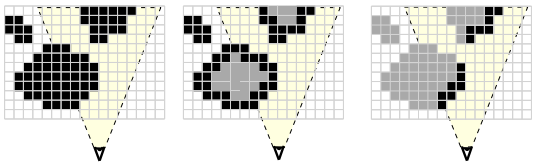


Figure 3: Voxel volume (left), shell-like boundary voxels (middle), culling invisible voxels (right).

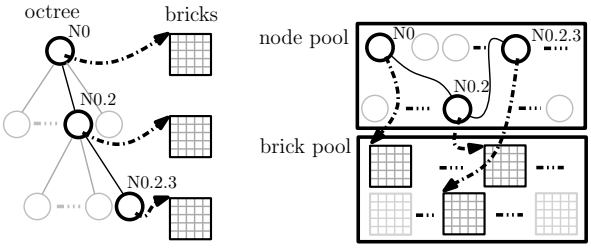


Figure 4: Sketch of the GigaVoxel data structure [7].

Laine and Karras [25] store an octree data structure in a single pool. The data is divided into blocks which represent contiguous areas of memory allowing local addressing and taking advantage of fast memory access operations on the GPU (caching). The data structure is designed to compactly store mesh-based scenes but their open-source implementation could probably be adapted to represent cloud volumes.

Miller et al. [28] use a grid representation on the coarse scale and nest octrees within those cells. This provides fast grid marching on a large scale, and adaptive sampling on the small scale. A fixed data structure, resembling 4-level octrees, allows to directly access the octree's nodes without requiring pointers. However, their current implementation assumes that the whole scene fits into the memory of the graphics device, which limits the model size. A streaming-based data structure for dynamic volumes was announced as future work.

The real-time voxelization of scene geometry, as proposed by Forester et al. [12], allows to transform rasterizable geometry to an octree representation on-the-fly on the GPU, and thus to apply voxel-based lighting and rendering to a surface-based geometry.

Octrees are a flexible data structure, generally capable of representing all types of clouds and supporting efficient lighting and rendering techniques. However, since octrees usually take advantage of axis-aligned grids, they are not directly applicable in an efficient way to large-scale cloud scenes, but would have to be nested in the spherical shell or used with a ray caster that takes into account the curvature of the Earth.

2.1.2 Binary Space Partitioning (BSP)

BSP, e.g., in form of kd-trees, recursively subdivides the space into half-spaces, concentrating at regions with high geometric detail and removing empty space. BSP could be used for cloud volume representation as well.

2.1.3 Bounding Volume Hierarchies (BVH)

BVH enclose scene geometry by bounding planes. Recently, BVH were used for structuring particle systems [14], which could also be employed for cloud volumes.

2.2 Implicit Representations

A common way to model and represent a cloud's density field is the use of procedural methods. While the

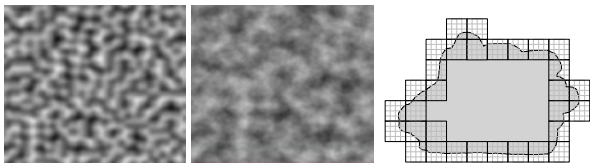


Figure 5: Perlin noise [31] and "fractal sum" [11] functions (left). A hypertexture applied to a surface (right).

overall structure is usually specified by simple geometric primitives, like spheres or ellipsoids, the internal, high-resolution structure is modeled by a function.

Perlin and Hoffert [31] introduce space-filling shapes, based on procedural functions, so-called *hypertextures*. Ebert et al. [11] propose many additional noise functions (see Fig. 5, left), and create various natural patterns applicable also for clouds.

The sampling of implicitly defined densities can become expensive and harm rendering performance. Schpok et al. [36] propose to evaluate the procedural functions on-the-fly on the GPU during the rendering by using a fragment shader program and a 3D texture containing Perlin noise.

Kniss et al. [22] use procedural functions for geometric distortion which adds a fractal appearance to regular shapes by changing the vertex positions of the geometry rendered. They apply this distortion during the rendering process by using vertex shader programs.

Bouthors et al. [4] use hypertextures in combination with surface-bounding volumes for creating a fractal boundary appearance; see Fig. 5, right.

Implicitly specifying a volume via procedural functions is a compact volume representation. The computational cost can be mitigated by employing parallel processing on the GPU and taking advantage of hardware-supported tri-linear interpolation of 3D textures. Procedural techniques are perfect for clouds with a fractal boundary appearance. However, they only provide the fine-scale volume structure while the overall cloud shape has to be modeled by other techniques.

2.3 Particle Systems

The volume is represented by a set of particles with a pre-defined volume. Usually spherical particles with a radial density function are used.

Nishita et al. [30] promote the use of particles with a Gaussian density distribution, so-called *metaballs*. While the metaballs in [30] are used only to create a volume density distribution, Dobashi et al. [10] directly light and render the metaballs and visualize medium-size cloud scenes of Cumulus-like clouds with a diffuse boundary appearance.

Bouthors and Neyret [2] use particles to create a shell-like volume representation for Cumulus-like clouds. Their algorithm iteratively places particles at the interface of a cloud volume, with smaller particles being placed upon the interface of larger particles. This creates a hierarchy of particles with decreasing radius and specifies the cloud's surface, which can be transformed, e.g., to a triangle mesh.

Efficient transformation algorithms were developed to match the favored lighting and rendering approaches. Cha et al. [6] transform the density distribution given by a particle system to a regular grid by using the GPU, while Zhou et al. [44] propose the inverse process transforming a density field to *radial basis function* (RBF) representation. This low-resolution particle system is accompanied by a high-resolution grid, which stores deviations of the particles' density distribution from the initial density field in a so-called *residual field*. *Perfect spatial hashing* allows a compact storage of this field.

Particles systems are a compact volume representation and directly support many cloud modeling techniques. Spherical particles are well suited for Cumulus-like clouds or dense cloud volumes, but less appropriate for stratified cloud layers with a large horizontal extension or for thin, fiber-like clouds.

2.4 Surface-Bounded Volumes

The cloud volume is represented by its enclosing hull, usually given as a triangle mesh. Since no information on the internal structure is available, usually a homogeneous volume is assumed.

Bouthors et al. [4] demonstrate the use of surface-bounded volumes for visualizing single Cumulus clouds. A triangle mesh is used in combination with a hypertexture to add small-scale details at the boundaries. A sophisticated lighting model reproduces a realistic appearance (see Sec. 4.1.4).

Porumbescu et al. [32] propose *shell maps* to create a volumetric texture space on a surface. A tetrahedral mesh maps arbitrary volumetric textures to this shell.

Surface-bounded volumes are a very compact representation of cloud volumes, allowing for efficient rendering and for incorporating sophisticated lighting techniques. However, they are only applicable if a quick saturation of a ray entering the volume may be assumed. While this is valid for dense clouds it does not apply to thin or layered clouds with their optically thin boundaries. Also special rendering techniques have to be developed for allowing perspectives from within the clouds.

2.5 Clouds as Layers

Clouds represented by a single layer, usually rendered as a textured triangle mesh, allow fast rasterization-based rendering and are the traditional technique to present clouds, e.g., during weather presentations [24].

Bouthors et al. [3] visualize cloud layers viewed from the ground or from above. They achieve a realistic appearance of the cloud by applying a sophisticated, viewpoint-dependent lighting model.

The 2D representation is especially predestined for thin cloud sheets viewed from the ground, or for visualizing the cloud tops viewed, e.g., from the perspective of a satellite. However, this non-volumetric representation limits the possible perspectives and generally does not allow for animations, like transitions of the viewpoint from space to the ground, or cloud fly-throughs.

3 CLOUD RENDERING TECHNIQUES

3.1 Rasterization-based Rendering

3.1.1 Volume Slicing

Volume slicing is a straightforward method for rendering regular grids. The slices are usually axis aligned and rendered in front-to-back order (or vice-versa), applying viewing transformations and blending. While volume slicing is not necessarily a rasterization-based method, most algorithms exploit the highly optimized texturing capabilities of the graphics hardware.

Schpok et al. [36] use volume slicing for cloud rendering and propose the use of the GPU also for adding detailed cloud geometry on-the-fly by using a fragment shader program. This reduces the amount of data which has to be transferred onto the GPU to a low-resolution version of the cloud volume. Harris et al. [15] create the density volume by a CFD simulation on the GPU for creating a 3D density and light texture which can efficiently be rendered on the GPU. Sparing the transfer of the volumetric data from the CPU, they achieve interactive frame rates for small volumes (up to 64^3), creating soft, diffuse clouds. Hegeman et al. [17] also use 3D textures to capture the volume density and the pre-calculated source radiance, and evaluate a lighting model in a CG shader program during rendering.

Zhou et al. [44] visualize smoke represented by a low-resolution particle system accompanied by a high-resolution density grid, stored in compressed form (see Sec. 2.3). During the rendering process the lighted particles are first rasterized and converted to a 3D texture by applying parallel projection rendering along the z-axis to fill slices of the 3D texture. Thereby, for each slice, all particles are traversed and, in case of intersection with the slicing plane, rendered as textured quads. During this pass also the residual field is evaluated and stored in a separate 3D texture. In a second step perspective rendering is applied by slicing in back-to-front manner. Again, for each slice all particles are traversed and bounding quads are rendered triggering a fragment shader which composes the light and density information from the 3D textures. They achieve interactive frame rates under dynamic lighting conditions for moderate volume sizes which completely fit into the GPU's memory (128^3 in their examples), based on 0.5 to 1.5 hours of pre-processing time.

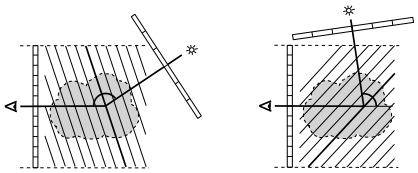


Figure 6: Volume rendering by half-angle slicing.

Half-Angle Slicing: Slicing the volume at planes oriented halfway between the lighting and viewing direction (or its inverse, see Fig. 6) is called half-angle slicing. It allows to combine the lighting and rendering of the volume in a single process by iterating once over all slices. During this single-volume pass two buffers are maintained and iteratively updated: one for accumulating the attenuation of radiance in the light direction, and one for accumulating the radiance for the observer (usually in the frame buffer). Due to the single pass through the volume, the lighting scheme is limited to either forward or backward scattering.

Kniss et al. [22] use half-angle slicing in combination with geometric distortion, modifying the geometry of shapes on-the-fly during rendering (see Sec. 2.2) by using vertex shader programs. Riley et al. [35] visualize thunderstorm clouds, taking into account different scattering properties of the volume.

For avoiding slice-like artifacts volume slicing techniques have to employ small slicing intervals, resulting in a large number of slices which can harm rendering performance and introduce numerical problems. Inhomogeneous volumes, like in Figs. 1.2 or 1.14, would also require a huge number of slices for appropriately capturing the volume's geometry and avoiding artifacts in animations with a moving viewpoint. Slicing-based methods therefore seem to favor volumes with soft or diffuse boundaries and, thus, are applicable to clouds with sharp boundaries only to a limited extent.

3.1.2 Splatting

Splatting became the common method for rendering particle systems. Particles, which are usually specified as independent of rotation, can be rendered by using a textured quad representing the projection of the particle onto a plane, also called splat or footprint. The particles are rendered in back-to-front order, applying blending for semi-transparent volumes.

Dobashi et al. [10] use metaballs for splatting cloud volumes. Harris et al. [16] accelerate this approach by re-using impostors, representing projections of a set of particles, for several frames during fly-throughs. The use of different particle textures can enhance the clouds' appearance [42, 18].

Since usually a single color (or luminance) is assigned to each particle and the particles do not represent a distinct geometry, but a spherical, diffuse volume (due to the lack of self-shadowing within a particle), the splatting approach is limited to visualizing clouds with a soft, diffuse appearance. Clouds with a distinct surface geometry, like Cumulus clouds, cannot be reproduced realistically.

3.1.3 Surface-Based Volume Rendering

Nowadays, rendering clouds as textured ellipsoids is no more regarded as realistic. The same applies to the surface-bounded clouds of [41]. A triangle mesh is created by surface subdivision of an initial mesh, created by a marching cubes algorithm applied on weather forecast data. The rendering of the semi-transparent mesh, however, is prone to artifacts on the silhouette.

Bouthors et al. [4] resurrected surface-based cloud rendering by using fragment shader programs which calculate the color for each fragment of a triangle mesh at pixel-basis. This allows to apply a sophisticated volume lighting scheme and the sampling of a hypertexture superposed onto the surface on-the-fly for each pixel. They achieve a realistic, real-time visualization of dense clouds with fractal and sharp boundaries.

3.2 Ray Casting-Based Rendering

3.2.1 Ray Marching

Ray marching casts rays into the scene and accumulates the volume densities at certain intervals. For rendering participating media volumes, like clouds, the illumination values of the volume have to be evaluated, either by applying a volume lighting model on-the-fly or by retrieving the illumination from a pre-computed lighting data structure.

Grids: Cha et al. [6] transform a particle-based volume representation to a regular density grid for applying ray marching. The volume density is sampled within a 3D texture, combined with the illumination, stored in a separate 3D texture, and accumulated along the viewing rays. Geometric detail is added to the low-resolution volume representation on-the-fly by slightly distorting the sampling points of the ray march according to a precomputed 3D procedural noise texture. For volumes fitting into the memory of the GPU (around 256^3 in their examples), they achieve rendering times of a few seconds (including the lighting calculation).

BVHs: A particle system stored within a kd-tree structure is proposed by Gourmel et al. [14] for fast ray tracing. The technique could be used for cloud rendering, e.g., by ray marching through the particle volumes and adding procedural noise as proposed in [4] or [6].

Octrees: The huge amount of data caused by volumetric representations can be substantially reduced by using ray-guided streaming [7, 13, 25]. The GigaVoxel algorithm [7] is based on an octree with associated bricks (Sec. 2.1.1) which are maintained in a pool and streamed on demand. The bricks at different levels of the octree represent a multi-resolution volume, similar to a mip-map texture. Sampling densities at different mip-map resolutions simulates cone tracing (see Fig. 4, right). The octree is searched in a stack-less manner, always starting the search for a sampling position at the root node. This supports the cone-based sampling of the volume since the brick values at all levels can be collected during the descent. The implementation employs shader programs on the GPU, and achieves 20–90 fps, with volumetric resolutions up to 160k.

4 LIGHTING TECHNIQUES

4.1 Participating Media Lighting

We review recent volume lighting approaches and refer to [5] for a survey of traditional, mainly off-line lighting models.

4.1.1 Single-Scattering Approximation

Slice-Based: Schpok et al. [36] use volume slicing for creating a low-resolution volume storing the source radiances. This so-called light volume is oriented such that light travels along one of the axes, which allows a straightforward light propagation from slice to slice. The light volume storing the source radiance is calculated on the CPU and transferred to a 3D texture on the GPU for rendering.

In the half-angle slicing approach by Kniss et al. [22, 23] the light is propagated from slice to slice through the volume by employing the GPU's texturing and blending functionalities. A coarser resolution can be used for the 2D light buffer (Fig. 6), thus saving resources and accelerating the lighting process.

Particle-Based: Dobashi et al. [10] use shadow casting as a single-scattering method for lighting a particle system. The scene is rendered from the perspective of the light source, using the frame buffer of the GPU as a shadow map. The particles are sorted and processed in front-to-back manner relative to the light source. For each particle the shadow value is read back from the frame buffer before proceeding to the next particle and splatting its shadow footprint. This read-back operation forms the bottleneck of the approach which limits either the model size or the degree of volumetric detail. Harris and Lastra [16] extend this approach by simulating multiple forward scattering and propose several lighting passes for accumulating light contributions from different directions including skylight.

Bernabei et al. [1] evaluate for each particle the optical depth towards the boundary of the volume for a set

of directions and store it in a spherical harmonic representation. Each particle therefore provides an approximate optical depth of the volume for all directions, including the viewing and lighting directions (Fig. 8, left). The rendering process reduces to accumulating the light contributions of all particles intersecting the viewing ray, thus sparing a sorting of the particles, and provides interactive frame rates for static volumes. For the source radiance evaluation a ray marching on an intermediate voxel-based volume is used in an expensive pre-process. Zhou et al. [44] accomplish this in real-time by employing spherical harmonic exponentiation; see Sec. 4.1.6.

4.1.2 Diffusion

Light distribution as a diffusion process is a valid approximation in optically thick media, but not in inhomogeneous media or on its boundary. Therefore Max et al. [27] combine the diffusion with anisotropic scattering and account for cloudless space to reproduce, e.g., silver-lining. However, the computational cost is still high and causes rendering times of several hours.

4.1.3 Path Tracing

Path tracing (PT) applies a Monte Carlo approach to solve the rendering equation. Hundreds or even thousands of rays are shot into the scene for each pixel and traced until they reach a light source. PT produces unbiased, physically correct results but generally suffers from noise and low convergence rates. Only recent acceleration techniques allow an efficient rendering of large scenes, at least for static volumes.

In [39, 43], the volume sampling process of PT is accelerated by estimating the free path length of a medium in advance and by using this information for a sparse sampling of the volume ("*woodcock tracking*"). While Yue et al. [43] use a kd-tree for partitioning the volume, Szirmay-Kalos et al. [39] use a regular grid.

4.1.4 Path Integration

Path integration (PI) is based on the idea of following the path with the highest energy contribution and estimates the spatial and angular spreading of light along this so-called *most probable path* (MPP). PI favors high-order scattering with small scattering angles, and tends to underestimate short paths, diffusive paths and backward scattering.

The initial calculation model of Premoze et al. [33] is based on deriving a *point spread function* which describes the blurring of incident radiance. Hegeman et al. [17] proposed the evaluation of this lighting model using a shader program executed on the GPU. They visualize dynamic smoke (at resolutions up to 128^3) within a surface-based scene at interactive frame rates.

Bouthors et al. [4] use PI in combination with pre-computed tables which are independent of the cloud volume. In an exhaustive pre-computation process the impulse response along the MPP at certain points in a slab is calculated and stored as spherical harmonics (SH) coefficients (Fig. 7, left). During the rendering process the slabs are adjusted to the cloud volume (Fig. 7, right), and used for looking up the pre-computed light attenuation values. Multiplying it with the incident radiance at the cloud surface around the so-called collector area yields the resulting illumination. The model is implemented as a shader program which evaluates the lighting model pixel-wise in real-time.

4.1.5 Photon Mapping

Photon mapping (PM) traces photons based on Monte Carlo methods through a scene or volume and stores them in a spatial data structure that allows a fast nearest-neighborhood evaluation (usually a kd-tree). However, when applied to volumes the computational and memory costs are significant.

Cha et al. [6] combine photon tracing with irradiance caching, accumulating the in-scattered radiances at voxels of a regular grid. The gathering process reduces to a simple ray marching in the light volume executed on the GPU. This provides rendering times of a few seconds for medium-size volumes.

Progressive photon mapping as proposed by Knaus and Zwicker [21] improves traditional static photon mapping (PM) by reducing the memory cost and can also be applied to participating media. Photons are randomly traced through the scene, but instead of storing them in a photon map, they are discarded and the radiances left by the photons are accumulated in image space. Jarosz et al. [20] combine progressive PM with an improved sampling method of photon beams.

4.1.6 Mixed Lighting Models

Lighting techniques can be combined by evaluating different components of the resulting light separately.

Particle-Based: Zhou et al. [44] evaluate the source radiances at the particles' centers by accumulating the light attenuation of all particles onto each particle. Thereto a convolution of the background illumination function with the light attenuation function of the particles (spherical occluders) is applied (Fig. 8,

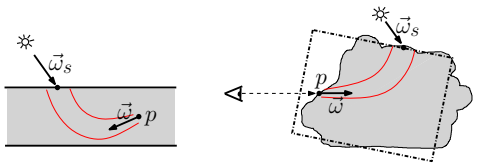


Figure 7: Pre-computed light in a slab (left), fitting slabs to a cloud surface (right).

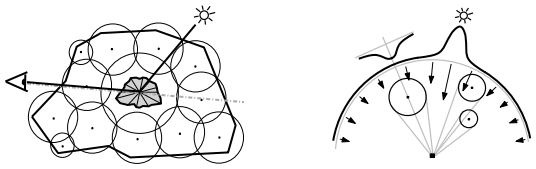


Figure 8: Pre-computed optical density at a particle center (left). Convolution of background illumination with the light attenuation by spherical occluders (right).

right). The functions are represented by low-order spherical harmonics (SH) and the convolution is done in logarithmic space where the accumulation reduces to summing SH coefficients; a technique called SH exponentiation (SHEXP), introduced by Ren et al. [34] for a shadow calculation. For multiple scattering an iterative diffusion equation solver is used on the basis of the source radiance distribution. This includes solving linear systems of dimension n, with n being the number of particles. The approximation of the smoke density field by a small number of particles (around 600 in their examples) allows interactive frame rates under dynamic lighting conditions. However, the super-quadratic complexity in terms of the particle number prohibits the application to large volumes.

Surface-Based: Bouthors et al. [3] separately calculate the single and multiple-scattering components of light at the surface of a cloud layer. The single-scattering model reproduces silver lining at the silhouettes of clouds and back-scattering effects (glories and fog-bows). Multiple-scattering is evaluated by using a BRDF function, pre-calculated by means of path tracing. Inter-reflection of the cloud layer with the ground and sky is taken into account by a radiosity model. They achieve real-time rendering rates by executing the lighting model on the GPU as a shader program.

4.1.7 Heuristic Lighting Models

Neyret [29] avoids costly physical simulations when a-priori knowledge can be used to simulate well-known lighting effects. He presents a surface-based shading model for Cumulus clouds that simulates inter-reflection in concave regions (light traps) and the corona at the silhouette.

Wang [42] does not employ a lighting model at all, but lets an artist associate the cloud's color to its height.

4.1.8 Acceleration Techniques

Pre-Computed Radiance Transfer: The lighting distribution for static volumes under changing lighting conditions can be accelerated by pre-computing the radiance transfer through the model.

Lensch et al. [26] pre-compute the impulse response to incoming light for a mesh-based model. Local lighting effects are simulated by a filter function applied to a light texture. Looking up the pre-computed vertex-to-vertex throughput yields global lighting effects.

Sloan et al. [38] use SH as emitters, simulate their distribution inside the volume and store them as transfer vectors for each voxel. The volume can be rendered efficiently in combination with a shader-based local lighting model under changing light conditions.

Radiance Caching: Jarosz et al. [19] use radiance caching to speed up ray marching rendering process. Radiance values and radiance gradients are stored as SH coefficients within the volume and re-used.

4.2 Global Illumination

Inter-cloud shadowing and indirect illumination cannot be efficiently simulated by participating media lighting models. These effects require global illumination techniques, usually applied in surface-based scenes.

4.2.1 Shadows

Sphere-based scene representations allow a fast shadow calculation, either, e.g., by using SH for representing the distribution of blocking geometries [34] (Fig. 8, right), or by accumulating shadows in image space [37].

The CUDA implementation of the GigaVoxel algorithm [8] allows to render semi-transparent voxels and to cast shadow rays. Employing the inherent cone tracing capability produces soft shadows.

4.2.2 Indirect Illumination

Voxel cone tracing applied to a sparse octree on the GPU is used by Crassin et al. [9] for estimating indirect illumination. The illumination at a surface point is evaluated by sampling the neighborhood along a small number of directions along cones.

A fast ray-voxel intersection test for an octree-based scene representation is used in Thiedemann et al. [40] for estimating near-field global illumination.

5 SUMMARY AND OUTLOOK

In the following table we summarize efficient lighting and rendering techniques for clouds with different types of volume (rows) and boundary appearances (columns).

	diffuse	fractal	sharp
dense	[6, 7, 10, 15] [16, 35, 44]	[4, 7, 11] [17, 22]	[4, 42]
thin	[22, 36, 44]	[11, 17, 36]	do not exist
layer	[3]		

The approaches surveyed represent a set of impressive but specialized solutions that employ fairly diverse techniques. However, all known approaches cover only some of the phenomena found in nature (see Sec. 1.1). Extensive memory usage or the algorithm's complexity often limit the size of a cloud volume. Thus, the realistic visualization of real-world cloud scenes still is and will remain widely open for future research for years to come.

Major future challenges to be tackled include the efficient handling of

- heterogeneous cloud scene rendering, i.e., the simultaneous visualization of different cloud types, each favoring a specific cloud representation, lighting and rendering technique;
- large-scale cloud scenes, e.g., clouds over Europe;
- cloud-to-cloud shadows and cloud inter-reflection, i.e., the combination of global illumination techniques with participating media rendering;
- the inclusion of atmospheric scattering models for lighting clouds;
- cloud rendering at different scales, i.e., views of clouds from close and far, and seamless transitions in between, requiring continuous LOD techniques;
- temporal cloud animations implying dynamic volumes and employing cloud simulation models.

ACKNOWLEDGEMENTS

Work supported by Austrian FFG Grant #830029.

6 REFERENCES

[1] D. Bernabei, F. Ganovelli, N. Pietroni, P. Cignoni, S. Pattanaik, and R. Scopigno. Real-time single scattering inside inhomogeneous materials. *The Visual Computer*, 26(6-8):583–593, 2010.

[2] A. Bouthors and F. Neyret. Modeling clouds shape. In *Eurographics, Short Presentations*, 2004.

[3] A. Bouthors, F. Neyret, and S. Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *EG Workshop on Natural Phenomena*, Vienna, Austria, 2006. Eurographics.

[4] A. Bouthors, F. Neyret, N. Max, É. Bruneton, and C. Crassin. Interactive multiple anisotropic scattering in clouds. In *Proc. ACM Symp. on Interactive 3D Graph. and Games*, 2008.

[5] E. Cerezo, F. Perez-Cazorla, X. Pueyo, F. Seron, and F. X. Sillion. A survey on participating media rendering techniques. *The Visual Computer*, 2005.

[6] D. Cha, S. Son, and I. Ihm. GPU-assisted high quality particle rendering. *Comp. Graph. Forum*, 28(4):1247–1255, 2009.

[7] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graph. and Games (I3D)*, Boston, MA, Etats-Unis, February 2009. ACM, ACM Press.

[8] C. Crassin, F. Neyret, M. Sainz, and E. Eisemann. *GPU Pro*, chapter X.3: Efficient Rendering of Highly Detailed Volumetric Scenes with GigaVoxels, pages 643–676. AK Peters, 2010.

[9] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. Interactive indirect illumination using voxel cone tracing. *Comp. Graph. Forum (Proc. Pacific Graph. 2011)*, 30(7), 2011.

[10] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Comp. Graphics (SIGGRAPH '00 Proc.)*, pages 19–28, 2000.

[11] D. S. Ebert, F. Musgrave, P. Peachey, K. Perlin, and S. Worley. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 3rd edition, 2003.

[12] V. Forest, L. Barthe, and M. Paulin. Real-time hierarchical binary-scene voxelization. *Journal of Graphics, GPU, & Game Tools*, 14(3):21–34, 2009.

[13] E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24(7-9):797–806, 2008. Proc. CGI 2008.

[14] O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin. Fitted BVH for fast raytracing of metaballs. *Comp. Graph. Forum*, 29(2):281–288, May 2010.

[15] M. J. Harris, W. V. Baxter, Th. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proc. SIGGRAPH/Eurographics '03 Conference on Graph. Hardware*, pages 92–101. Eurographics Association, 2003.

[16] M. J. Harris and A. Lastra. Real-time cloud rendering. *Comput. Graph. Forum (Proc. IEEE Eurographics '01)*, 20(3):76–84, 2001.

[17] K. Hegeman, M. Ashikhmin, and S. Premože. A lighting model for general participating media. In *Proc. Symp. on Interactive 3D Graph. and Games (I3D '05)*, pages 117–124, New York, NY, USA, 2005. ACM.

[18] R. Hufnagel, M. Held, and F. Schröder. Large-scale, realistic cloud visualization based on weather forecast data. In *Proc. IASTED Int. Conf. Comput. Graph. and Imaging (CGIM'07)*, pages 54–59, February 2007.

[19] W. Jarosz, C. Donner, M. Zwicker, and H. W. Jensen. Radiance caching for participating media. *ACM Trans. Graph.*, 27(1):1–11, 2008.

[20] W. Jarosz, D. Nowrouzezahrai, R. Thomas, P. P. Sloan, and M. Zwicker. Progressive photon beams. *ACM Trans. Graph. (SIGGRAPH Asia 2011)*, 30(6):181:1–181:12, December 2011.

[21] C. Knaus and M. Zwicker. Progressive photon mapping: A probabilistic approach. *ACM Trans. Graph.*, 30(3):25:1–25:13, May 2011.

[22] J. M. Kniss, S. Premože, Ch. D. Hansen, and D. S. Ebert. Interactive translucent volume rendering and procedural modeling. In *Proc. IEEE Visualization '02*, pages 109–116. IEEE Comput. Society Press, 2002.

[23] J. M. Kniss, S. Premože, Ch. D. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Trans. Visualiz. Comput. Graph.*, 9(2):150–162, 2003.

[24] H.-J. Koppert, F. Schröder, E. Hergenröther, M. Lux, and A. Trembilski. 3d visualization in daily operation at the dwd. In *Proc. ECMWF Worksh. on Meteorolog. Operat. Syst.*, 1998.

[25] S. Laine and T. Karras. Efficient sparse voxel octrees.

In *Proc. ACM SIGGRAPH Symp.on Interactive 3D Graph.and Games (I3D '10)*, pages 55–63, New York, NY, USA, 2010. ACM.

[26] H. P. A. Lensch, M. Goesele, Ph. Bekaert, J. Kautz, M. A. Magnor, J. Lang, and H.-P. Seidel. Interactive rendering of translucent objects. In *Proc. Pacific Conf. on Comp. Graph. and Appl. (PG '02)*, pages 214–. IEEE Computer Society, 2002.

[27] N. Max, G. Schussman, R. Miyazaki, and K. Iwasaki. Diffusion and multiple anisotropic scattering for global illumination in clouds. *Journal of WSCG 2004*, 12(2):pp. 277, 2004.

[28] A. Miller, V. Jain, and J. L. Mundy. Real-time rendering and dynamic updating of 3-d volumetric data. In *Proc. Worksh. on General Purpose Process. on Graph. Process. Units (GPGPU-4)*, pages 8:1–8:8, New York, NY, USA, 2011. ACM.

[29] F. Neyret. A phenomenological shader for the rendering of cumulus clouds. Technical Report RR-3947, INRIA, May 2000.

[30] T. Nishita, Y. Dobashi, and E. Nakamae. Display of clouds taking into account multiple anisotropic scattering and skylight. *Comput. Graphics (SIGGRAPH '96 Proc.)*, pages 379–386, 1996.

[31] K. Perlin and E. M. Hoffert. Hypertexture. In *Comp. Graph. (SIGGRAPH '89 Proc.)*, volume 23 (3), pages 253–262, July 1989.

[32] S. D. Porumbescu, B. Budge, L. Feng, and K. I. Joy. Shell maps. *ACM Trans. Graph.*, 24:626–633, July 2005.

[33] S. Premože, M. Ashikhmin, R. Ramamoorthi, and Sh. K. Nayar. Practical rendering of multiple scattering effects in participating media. In *Proc. Eurographics Worksh. on Render. Tech.*, pages 363–373. Eurographics Association, June 2004.

[34] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P. P. Sloan, H. Bao, Q. Peng, and B. Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *ACM Trans. Graph. (SIGGRAPH '06 Proc.)*, pages 977–986, 2006.

[35] K. Riley, D. S. Ebert, Ch. D. Hansen, and J. Levit. Visually accurate multi-field weather visualization. In *Proc. IEEE Visualization (VIS'03)*, pages 37–, Washington, DC, USA, 2003. IEEE Computer Society.

[36] J. Schpok, J. Simons, D. S. Ebert, and Ch. D. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proc. SIGGRAPH/Eurographics '03 Symp. Computer Anim.*, pages 160–166. Eurographics Association, July 2003.

[37] P. P. Sloan, N. K. Govindaraju, D. Nowrouzezahrai, and J. Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proc. Pacific Conf. on Comp. Graph. and Appl.*, pages 97–105, Washington, DC, USA, 2007. IEEE Computer Society.

[38] P. P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Trans. Graph. (SIGGRAPH '02 Proc.)*, pages 527–536, July 2002.

[39] L. Szirmay-Kalos, B. Tóth, and M Magdics. Free path sampling in high resolution inhomogeneous participating media. *Comp. Graph. Forum*, 30(1):85–97, 2011.

[40] S. Thiedemann, N. Henrich, Th. Grosch, and St. Mueller. Voxel-based global illumination. In *ACM Symp. on Interactive 3D Graph. and Games (I3D)*, 2011.

[41] A. Trembilski and A. Broßler. Surface-based efficient cloud visualisation for animation applications. *Journal of WSCG*, 10(1–3), 2002.

[42] N. Wang. Realistic and fast cloud rendering in computer games. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1, New York, NY, USA, 2003. ACM.

[43] Y. Yue, K. Iwasaki, B. Y. Chen, Y. Dobashi, and T. Nishita. Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media. *ACM Trans. Graph. (SIGGRAPH Asia 2010)*, 29:177:1–177:8, December 2010.

[44] K. Zhou, Z. Ren, St. Lin, H. Bao, B. Guo, and H. Y. Shum. Real-time smoke rendering using compensated ray marching. *ACM Trans. Graph.*, 27(3):36:1–36:12, August 2008.

Interactive Segmentation of Volume Data Using Watershed Hierarchies

Michal Hučko
Comenius University, Slovakia
michal.hucko@fmph.uniba.sk

Miloš Šrámek
Austrian Academy of Sciences, Austria
milos.sramek@oeaw.ac.at

ABSTRACT

The available methods for volume data segmentation and/or classification differ in the amount of the required user input on the one side and precision and ability to tweak the obtained results on the other. Automation of the task is more difficult when a general case is considered. In this paper we present an interactive segmentation and classification tool for arbitrary volumetric data, which is based on pre-segmentation of the volume in a hierarchy of homogeneous regions. The hierarchical subdivision allows for interactive adaptation of scale and precision according to the user requirements. The data is processed in three dimensions which minimises the amount of the needed interaction and gives instant overview of the resulting segmentation.

Keywords

Segmentation, user interface, watershed, scale-space.

1 INTRODUCTION

Segmentation methods, which are commonly used with volume data, can be classified in two groups – general methods and model-based methods [PB07]. Unlike general ones, model-based methods are based on certain knowledge about the target objects in the data as, for example, the expected object shape, mean voxel intensity, etc. In this paper we omit these as our goal is to provide a general segmentation tool which can be used with any volume data to segment arbitrary objects.

As the general methods use no additional information about the data which would aid in the process of segmentation, they require a greater amount of user interaction either in the form of process control, specification of parameters tailored to the current task or post-processing of the result. Our aim is to minimise this interaction while still leaving full control of the segmentation process to the user.

2 RELATED WORK

Common approach in volume data segmentation consists of selection of a region or object of interest in slices of the volume. The most basic general segmentation method used is manual segmentation where a user delineates the object of interest in the slices by hand. Although it is applicable at all times, its heavy user interaction demands are apparent. To speed up delineation of contours the LiveWire method [MMBU92] may be used. To obtain the desired results, a suitable cost function has to be first specified. The LiveWire method speeds up the process of contour drawing if the object of interest is clearly separeted from the rest of the data. If this condition is not satisfied for the current task, difficulties in cost function specification arise resulting in slow downs – user intervention is required and the method is reduced to manual segmentation.

As commonly used data sets have rather large dimensions, performing the segmentation on each slice is tedious and time consuming. An option is to segment only certain slices and interpolate the contour in the in-between slices [SPoP00]. Instead of the interpolation of the contour one may interpolate LiveWire control points instead and let the system compute contour in intermediate slices from the interpolated control points [SOB]. Precision of the resulting segmentation is dependent on the used interpolation and also on the set of key-slices. Problems might arise when topology or shape of the contour change rapidly between slices. Validation and potential correction of the interpolation is necessary.

Another common segmentation method is thresholding where voxels with intensities in certain range are selected. This method can be easily applied to certain data where tissue types can be distinguished by intensity (e.g. bone tissue in CT data), but applications to other data modalities or tissue types may pose a problem. If the task is to separate various objects of the same

tissue type thresholding may be used for preprocessing to reject clearly non-target voxels and thus localise the area of interest.

Two methods based on detection of homogeneous regions instead of contours are worth mentioning. General region growing requires user specified seed points and a suitable homogeneity criterion. Data is flooded from the seed points as long as the homogeneity criterion is satisfied, creating a homogeneous region. The watershed segmentation [VS91] is usually performed on a gradient image, which is treated as a topographic height map. Homogeneous regions with voxels with small gradient magnitude form valleys having voxels with high gradient magnitude on region borders as ridges. Although there exist various algorithms for computing the watershed transform, variants of two approaches are common – simulation of downhill water flow for each voxel or immersing of the relief into water. Exhaustive study of existing watershed algorithms is provided by Roedink and Meijster [RM00].

A broader overview and more detailed description of the existing segmentation methods can be found in [PB07].

2.1 Interaction techniques

All of the previously mentioned segmentation approaches differ in the way how a user can interact with the data to modify a partial result until the desired segmentation is achieved. For example, when delineating a contour the user directly sees the partial result and can undo recent steps if the contour starts to diverge from the desired position.

Thresholding requires numeric input – the threshold. Usually, a user is provided with a histogram from which the most suitable threshold value can be estimated. Clean separation of various tissue types based only on voxel intensity is rare and thus it may be difficult to find an optimal threshold value. Usage of other methods for refining the segmentation from thresholding (morphologic operations, connected component labeling, etc) is therefore convenient [STBH92].

Watershed segmentation produces a highly oversegmented result, especially if the data is spoiled by noise. Some methods allow merging of neighbouring regions if the shared border is weak (gradient magnitude is low). This situation is illustrated in figure 1 which shows gradient magnitude image of a CT head dataset slice – the corresponding watershed segmentation can be seen in figure 3 (red borders). The aim is to create segmentation in which the target object is labeled by a unique label. If this is not happening user intervention is usually required. For example, an interaction technique called marker-based watershed segmentation can be used [HP03], where a user specifies special include and exclude points in the data which prohibit merging of neighbouring regions if new region would contain markers of both types.

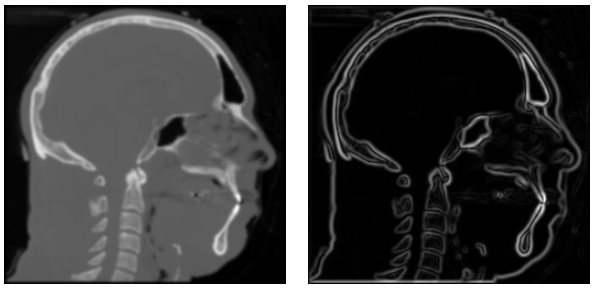


Figure 1: Slice from the Visible human male [Ack98] CT dataset with corresponding gradient magnitude image on right.

Watershed hierarchies [Beu94] were used in a technique based on interaction with slice views by Cates et al [CWJ05]. In such hierarchy, the order in which regions are merged defines a binary merge tree. Original regions form leaves and non-leaf nodes represent regions formed by merging of two regions – its children nodes. This tree may be used to segment an object from a data with a possibility to select large parts of the object by specification of high-positioned tree nodes and to refine the border by adding/removing low-positioned nodes.

In the paper by Armstrong et al [APB07] an extension of LiveWire or Intelligent Scissors called Live Surface was proposed. Data is presegmented into hierarchy of regions. Initial regions are computed using tobogganing [MB99] creating result equivalent to watershed segmentation (depends on the used watershed definition). For merging of the regions for higher levels in the hierarchy a special metric is used, which is based on the mean voxel intensity/colour and intensity/colour variance of a region. Segmentation is done by specifying two types of markers – inner and outer – which are then used to create a graph-cut in the region neighbourhood graph with minimal cost. Markers can be entered on an arbitrary cross section of the volume or directly in the 3D view allowing to add/remove parts to/from the border of an already segmented object.

3 THE SEGMENTATION TOOL

All of the segmentation approaches mentioned in the previous section were either proposed for two dimensional images or, if targeted to segmentation of 3D data, were used only for interaction in two dimensional space – on respective slices – or provided limited possibilities to modify the resulting segmentation directly in the 3D visualisation of the data. In our approach we let the user directly control the segmentation process by selecting fragments of the target object in 3D space. As manual segmentation by pixels/voxels is too cumbersome, data

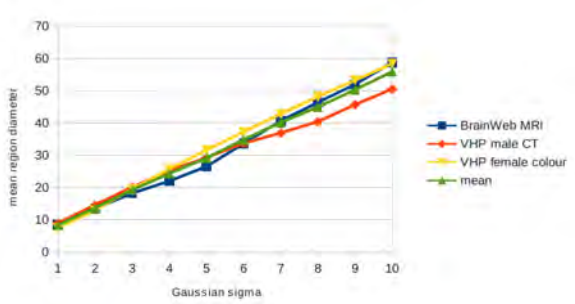


Figure 2: Measured dependency of mean region size on the Gaussian filter sigma. Three data sets were used – an MRI head, VHP male CT head and VHP female colour head dataset. Measured was diameter of minimal bounding sphere centered at the region's centre of mass.

is pre-segmented by the watershed transform as it creates homogeneous regions which can be used instead of single voxels.

3.1 Preprocessing

The watershed segmentation produces highly oversegmented results, because for each local minimum in the (usually gradient image of) original data a region is created. This can be a serious problem especially if noise is present in the data. To cope with this, data is usually first smoothed by the Gaussian filter. Smoothing, however, also shifts edges and removes weak edges, resulting in merging of regions which a user might desire in certain cases separated. Therefore, we perform the watershed segmentation on a sequence of derived data sets where each one is produced from the original by smoothing with a gradually increasing degree (increasing of the Gaussian sigma). As Koenderink showed [Koe84] this produces a set of images where on each image details smaller than certain size are increasingly ignored (the scale-space approach). As can be seen in figure 2, measurements on three different head datasets showed that dependency of mean bounding sphere diameter of regions on used Gaussian filter sigma can be approximated with function 1.

$$d(\sigma) = 5(\sigma - 1) + 10 \qquad (1)$$

To further reduce the starting number of regions, region merging based on mean region intensity or some other criterion can be used.

In order to correct position of the shifted edges, caused by smoothing, to the original position specified by the unsmoothed data or at least data at the lowest level of smoothing, an aligned region hierarchy is build. Spatial alignment of borders of corresponding regions on different hierarchy levels is achieved by the technique based on the maximum number of spatially overlapping voxels [SD02]. Thus, if later required, aligned regions

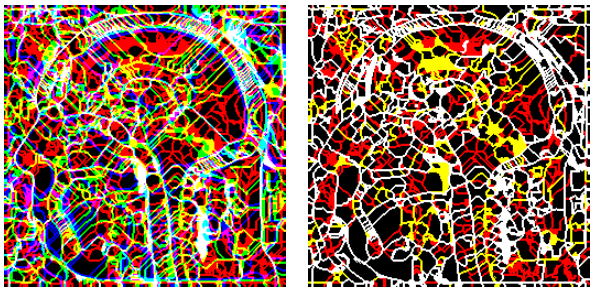


Figure 3: Three levels of watershed segmentation on visible human – male CT dataset. In the red channel are boundaries of regions for Gaussian smoothing with sigma 3, in green for sigma 5 and in blue for sigma 7. Overlapping boundaries are blended. On the left are the original watersheds, on the right the regions are aligned showing only red, yellow (red plus green) and white (all three levels) boundaries.

positioned higher in the hierarchy can be unambiguously decomposed to multiple smaller regions on some lower level.

As the last step of preprocessing the relevant neighbour information is extracted from the original voxel data and stored in a file. The described technique works equally well for scalar data (CT and MRI scans) as well as for multi-field data (dual energy CT, T1, T2 and PD MRI data etc). In the second case the watershed transform is performed on a gradient volume obtained as maximum of individual gradient fields.

3.2 GUI interaction

GUI of the segmentation application is shown in Figure 4. In this section all parts of the application as well as the segmentation workflow will be explained.

After the hierarchy is loaded into the application it is displayed in a tree widget allowing the user to navigate through it. Hovering or selecting a node highlights the corresponding region in a 3D visualisation window (Figure 5). This allows the user to choose an initial fragment of the region of interest. Depending on the target object and created hierarchy, regions on higher levels might consist of multiple target objects (e.g. various bones of skull). As pre-segmentation was performed at various scale levels, there is no need to repeat the process when the aforementioned problem arises – moving to lower levels in the hierarchy until the objects are separated is possible.

After the initial region is selected it is possible to display neighbouring regions which satisfy certain similarity criteria (Figure 6, also see section 3.3). The user can then select either one neighbour belonging to the target object by clicking into the 3D view or by selecting the neighbour in the list or can select all neighbors. Deselection of an undesired region is possible in a similar way, too. Once a neighbour is added/removed to/from

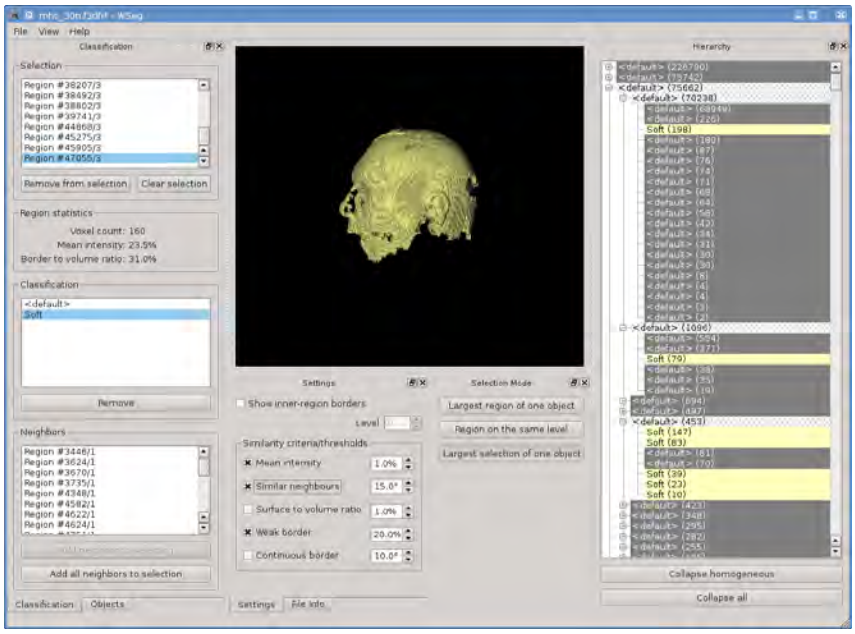


Figure 4: The segmentation tool: the region hierarchy is on the right. Left side from top to bottom: the list of selected regions, classification classes, the list similar neighbours of the regions in selection. At centre-bottom similarity criteria can be specified and above is the 3D visualisation and interaction window.

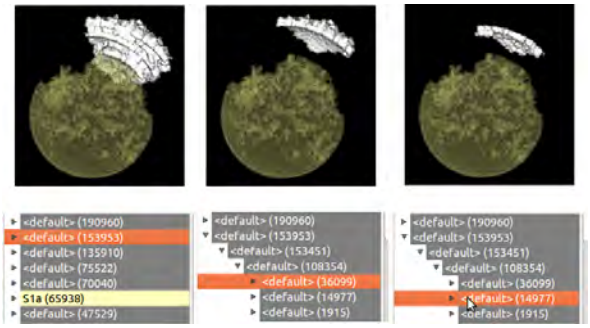


Figure 5: Selection of objects using the tree widget. The scene consist of concentric spheres/ellipsoids with different density and was intentionally spoiled by noise and superimposed low frequency density variation to make segmentation of the ellipsoids by thresholding impossible. Left: A region on the highest hierarchy level (white) was selected. One can see that it is composed of pieces of several ellipsoids. I we go down in the hierarchy (bottom row, in this case by two levels of the hierarchy), these pieces are separated and can be individually selected. The yellow region is a partially segmented and labeled part of the central sphere.

the selection, updated neighbours of the new selection are displayed. This allows traversal of the target object until all fragments are in the selection. Now classification class for the object can be created and all selected regions can be classified (Figure 6 right). Already classified regions are during selection refining displayed dimmed to provide context whereas the current selection is always highlighted.

3.3 Similarity criteria

To decide whether neighbouring regions are similar we implemented five different similarity criteria. Each criterion can be turned on or off and has a separate threshold value. When deciding whether two neighboring regions are similar all criteria which are turned on have to return a positive answer. Detailed description of the similarity criteria can be found below.

When searching for similar neighbours all neighbouring regions of the regions in the current selection are visited. As the selection might contain regions on different levels in the hierarchy we have to prevent multiple inclusion of same region – once directly and second time by including parent of the region. Naturally, regions positioned higher in the hierarchy are accepted in favor of the lower positioned child.

A second issue arises from the fact that we would like to move to a lower level of the hierarchy and to repeat the search for similar neighbors there, if we were not successful on a higher level. In this case, first, all regions in the selection are decomposed to regions on the lowest level – some lowest-level regions were directly present in the selection and some were selected indirectly by selection of its ancestor. Now we iterate through all pairs of the selected lowest-level regions and their neighbours (also at the lowest level). For both regions in the pair we ascend as many levels up in the hierarchy, until the original region in the selection which was decomposed to the currently processed lowest-level region is found. This leaves us with a list of pairs – (indirectly) selected region with its neighbour

– for each level not higher than level of the original selected region. Subsequently we start with the similarity tests. If a region and its neighbor on the higher level do not pass the selected similarity test we descent by one level and again perform the tests. By this higher positioned similar neighbours are found first.

A detailed description of the different similarity criteria follows:

Mean intensity Simple criterion comparing mean intensities of neighbouring regions. Difference in the intensities have to be in the specified interval for the test to pass. If the data contains multiple bands, the test must pass for all bands to be considered successful.

Similar neighbors In this criterion the prevalent directions of intensities in the region's neighborhood are compared. All neighbors of a region are visited and their center of mass is computed by averaging their geometric centroids weighted by their mean intensity. Subsequently, the direction vector, which points in the direction of growing intensity, is obtained by subtracting the center of mass from the geometric centroid of the region. Comparison of an angle between such vectors of two neighboring regions against a user defined threshold yields result of the similarity test.

Surface to volume ratio As working directly with the volume data would be slow and would increase memory requirements significantly, only derived information is used – mean intensity of a region, voxel count, etc. To compare region shapes ratio of the number of region's surface voxels to its total voxel count is used. The computed value is normalized to the $[0,1]$ range where 0 represents sphere-like objects and 1 string-like objects.

Weak borders For two regions to be similar in this criterion their border should have small mean gradient magnitude. Unlike the mean intensity criterion which compares mean values for whole regions, this criterion uses the original gradient data which were used during the creation of the most-detailed watershed segmentation. Intensity of the regions is n this criterion irrelevant.

Continuous border This criterion tries to find border in the data which spans multiple region boundaries. For two regions to pass this test they have to have a common neighbor. Both faces – first region with the common neighbor and second region with the common neighbor – have to be similar. For this, the angle between mean normal/gradient vectors of the faces is examined. Faces which are too small are ignored as their mean gradient vectors are based on too small set of values.

4 RESULTS

The application was implemented in C++ and uses OpenGL with GLSL. It was tested on the VHP dataset (the male head CT dataset). Individual bones of the skull, which are connected without a well defined border, were successfully segmented (figure 7 left). We also tested the application on an MRI scan of a human head. Figure 7 right shows segmentation of the brain cortex with certain gyri labeled as separate objects.

Preprocessing with three levels of watershed hierarchies and 3 additional levels of merging by density similarity took about 10 minutes. Both segmentations were produced in about 30 minutes.

In contrast to other methods, we intentionally omitted the possibility to interact through slices or cross sections to investigate the possibility for interaction only through the 3D view. If desired, display of the slices or cross sections can be easily added to the application, which may be then used for an initial selection.

If compared to traditional segmentation approaches, our method is most similar to region growing – assuming that mean intensity criterion is used. Neighbours to selected regions having similar intensity can be iteratively added to the selection until only regions with large difference in intensity remain. Because of the manual operation, user can omit regions which evaluate similar, but are not part of the target object. This can be essential when separating two or more objects of same or similar intensity, but different shape (as exemplified by segmentetion of the brain cortex in individual gyri). Automatic, or semi-automatic methods fail to separate these, if the interface is too weak or not present due to partial volume effect.

5 CONCLUSION

The presented segmentation and classification tool allows fast insight into data and fast segmentation of target structures while still leaving full control of the segmentation in user's hands. The amount of user interaction depends on data properties – resolution, presence of noise or other artefacts. Different similarity criteria were presented which should simplify and thus speed-up localisation of similar neighbouring regions in the data. Still, the user interaction is mainly done directly in the 3D visualisation window instead of slices. Operation in the visualisation window also gives direct visual feedback.

The concept was tested on the VHP dataset by segmentation of the bones of human skull and on MRI data by segmentation of respective gyri of the brain cortex.

6 ACKNOWLEDGEMENTS

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-20-056105, Science Grant Agency (VEGA, Slovakia)

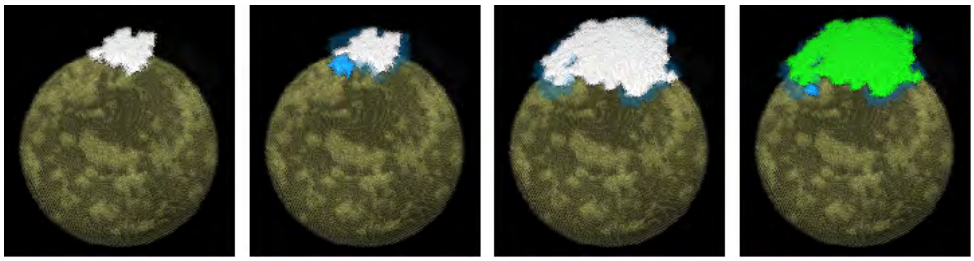


Figure 6: From left: selection of similar neighbors starting from an initial region. The blue regions are selection candidates (density similarity was used). White objects are already selected objects, the green color means that the object has already been labeled. The highlighted blue candidate object can be individually added to the selection or all candidates can be added at once.

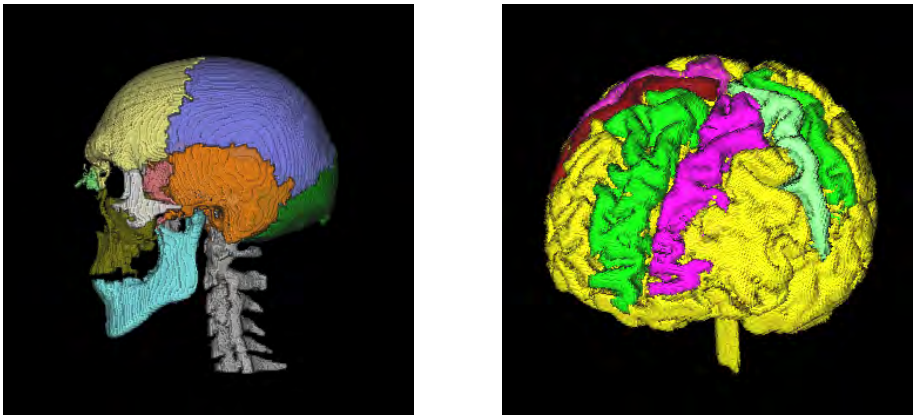


Figure 7: Left: segmentation of skull bones of the VHP male CT head dataset. Right: segmentation of brain and gyri in the MRI head dataset.

under contract No. 1/0631/11, FWF Austria under contract No. TRP-67-N23 and Comenius University under contract No. UK/229/2012. Miloš Šrámek is currently on leave at the the Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria.

7 REFERENCES

[Ack98] M.J. Ackerman, *The visible human project*, Proceedings of the IEEE **86** (1998), no. 3, 504–511.

[APB07] Christopher J. Armstrong, Brian L. Price, and William A. Barrett, *Interactive segmentation of image volumes with live surface*, Comput. Graph. **31** (2007), no. 2, 212–229.

[Beu94] S. Beucher, *Watershed, hierarchical segmentation and waterfall algorithm*, Mathematical morphology and its applications to image processing (J. Serra and P. Soille, eds.), Kluwer Academic Publishers, 1994, pp. 69–76.

[CWJ05] Joshua E. Cates, Ross T. Whitaker, and Greg M. Jones, *Case study: an evaluation of user-assisted hierarchical watershed segmentation.*, Medical Image Analysis (2005), 566–578.

[HP03] Horst K. Hahn and Heinz-Otto Peitgen, *Iwt-interactive watershed transform: a hierarchical method for efficient interactive and automated segmentation of multi-dimensional gray-scale images*, vol. 5032, SPIE, 2003, pp. 643–653.

[Koe84] J.J. Koenderink, *The structure of images*, Biological Cybernetics **50** (1984), no. 5, 363–370.

[MB99] E.N. Mortensen and W.A. Barrett, *Toboggan-based intelligent scissors with a four parameter edge model*, Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR '99), vol. II, 1999, pp. 452–458.

[MMBU92] Eric Mortensen, Bryan Morse, William Barrett, and Jayaram Udupa, *Adaptive boundary detection using Live-Wire two-dimensional dynamic programming*, IEEE Computers in Cardiology (1992), 635–638.

[PB07] B. Preim and D. Bartz, *Visualization in medicine: theory, algorithms, and applications*, The Morgan Kaufmann series in computer graphics, Morgan Kaufmann, 2007.

[RM00] Roerdink and Meijster, *The watershed transform: Definitions, algorithms and parallelization strategies*, FUNDINF: Fundamenta Informatica **41** (2000).

[SD02] M. Sramek and L. I. Dimitrov, *Segmentation of tomographic data by hierarchical watershed transform*, Journal of Medical Informatics and Technologies **3** (2002), 161–169.

[SOB] Zein Salah, Jasmina Orman, and Dirk Bartz, *Live-wire revisited*.

[SPoP00] Andrea Schenk, Guido Prause, and Heinz otto Peitgen, *Efficient semiautomatic segmentation of 3d objects in medical images*, In Proc. of Medical Image Computing and Computer-assisted Intervention (MICCAI, Springer, 2000, pp. 186–195.

[STBH92] T. Schiemann, U. Tiede, M. Bomans, and K. H. Höhne, *Interactive 3D-segmentation*, Visualization in Biomedical Computing 1992, Proc. SPIE 1808 (R. A. Robb, ed.), SPIE, Chapel Hill, NC, 1992, pp. 376–383.

[VS91] L. Vincent and P. Soille, *Watersheds in digital spaces: An efficient algorithm based on immersion simulations*, IEEE Transactions on Pattern Analysis and Machine Intelligence **13** (1991), 583–598.

Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field

Marcel Ritter
Institute of Basic Sciences in Civil Engineering[2]
University of Innsbruck
marcel.ritter@uibk.ac.at

Werner Benger
Center for Computation & Technology[1]
Institute for Astro- and Particle Physics[2]
werner@cct.lsu.edu
werner.benger@uibk.ac.at

ABSTRACT

Starting from the computation of a covariance matrix of neighborhoods in a point cloud, streamlines are utilized to reconstruct lines of linearly distributed points following the major Eigenvector of the matrix. This technique is similar to fiber tracking in diffusion tensor imaging (DTI), but in contrast is done mesh-free. Different weighting functions for the computation of the matrix and for the interpolation of the vector in the point cloud have been implemented and compared on artificial test cases. A dataset stemming from light detect and ranging (LIDAR) surveying served as a testbed for parameter studies where, finally, a power cable was reconstructed.

Keywords: tensor-field visualization; streamlines; mesh-free methods; particle systems; point cloud; covariance matrix; fiber tracking; LIDAR; DT-MRI

1 INTRODUCTION

Reconstructing lines from point clouds has an important application in light detection and ranging applications (LIDAR). The surveying of power lines and their geometrical analysis is of great interest for companies that transmit electrical energy. Large networks of electric facilities have to be maintained to guarantee stable electrical power supply and prevent power outages. LIDAR surveying is a suitable technique to either detect damages on the electrical facilities or detect high growing vegetation in power line corridors [19] [15]. We experiment on a new method to reconstruct linear structures, stemming from airborne LIDAR surveying. We utilize a method inspired by diffusion tensor imaging (DTI) fiber tracking developed, originally, for magnetic resonance imaging (MRI) to track neuronal structures in the human brain [5].

1.1 Related Work

Current algorithms for reconstructing power lines are usually based on data filtering followed by a segmentation of the filtered and reduced point cloud either directly on the point cloud data or on a rastered 2D image. Melzer [18] first computes a digital terrain model (DTM) by using the method by Kraus [14] to remove terrain points. The remaining points are projected onto a 2D gray-scale raster (image). A Hough-Transform (e.g. [11]) is utilized iteratively to detect straight lines. Later, Melzer [17] improved the segmentation of LIDAR data also for power cables, based on the so called mean shift clustering, originally developed for pattern recognition [9]. Liu et al. [16] introduced a methodology based on statistical analysis to first remove ground points. Then, they project points onto a 2D gray-scale raster (image) and do a Hough-Transform similar to Melzer [18], but use a different technique for the Hough-Transform [8] to detect straight lines. Jwa et al. [13] developed a four step method. First they select power-line candidates, by utilizing a voxel based Hough-Transform to recognize linear regions. After a filtering process they construct line segments based on geometric orientation rules and, finally, use a voxel-based piece-wise line detector to reconstruct the line geometries.

Weinstein et al. [23] worked on tracking linear structures in diffusion tensor data stemming from MRI. Besides following the major Eigenvector they developed some rules for overcoming areas of not linear diffusion. The flow of Eigenvectors was also used for segmentation and clustering in brain regions as, for example, shown in [6] and [20]. Jones discusses the study of connections in human brains. He states that tracking the

[1] Louisiana State University, Baton Rouge, LA-70803, USA
[2] University of Innsbruck, Technikerstraße 13/4, A-6020 Innsbruck, Austria

diffusion directions is still not solved a in stable way and is an active research area [12].

Our work is based on previous work on the direct visualization of the covariance matrix describing the local geometric properties of a neighborhood distribution within in a point cloud, the so called point distribution tensor [21].

1.2 Our Approach

In our method we do not want to remove any points but operate on the entire dataset to avoid artifacts due to a complex point removal method. Instead, we first compute the point distribution tensor for each point. Eigen-analysis of the tensor yields the major Eigenvector, which indicates the dominant orientation of a point distribution. We may follow this orientation by computing streamlines along this dominant Eigenvector field in regions where one Eigenvalue dominates, so-called linear regions. In contrast, regions where the points are distributed more isotropic, are indicated by the point distribution tensor's Eigenvalues becoming more similar values. We want to avoid these regions, as they will not correspond to power cables. This approach is very similar to the fiber-tracking approach in medical visualization, but in our case the integration of the Eigenvectors needs to be done in a mesh-free way, merely on a point distribution rather than uniform grids. Thus, it can be applied to airborne LIDAR data without resampling to uniform grids (which would reduce data resolution and introduce artifacts due to the chosen resampling method).

1.3 Overview of the Paper

Section 2 presents the mathematical background and describes the implementation of the algorithm in section 2.2. Section 2.3 shows verifications by means of simple artificial point distributions. Here, the influence of different weighting functions on the tensor computation and the vector field interpolation during streamline integration is investigated. Also, two different numerical integration schemes are tested. In section 3 one set of power cables is reconstructed from a LIDAR data set stemming from actual observations. We then explore the available parameter space for weighting and integration in order to identify the best values for the given scenario.

2 ALGORITHM

2.1 Background

In [21] we defined the "point distribution tensor" of a set of N points $\{P_i : i = 1,...,N\}$ as

$$S(P_i) = \frac{1}{N}\sum_{k=1}^{N} \omega_n(|t_{ik}, r|)(t_{ik} \otimes t_{ik}^{\tau}), \qquad (1)$$

whereby $\otimes$ denotes the tensor product, $^{\tau}$ the transpose and $t_{ik} = P_i - P_k$. $\omega_n(|t_{ik}|, r)$ is a weighting function dependent on the distance of a point sample to a center point P_i and a radius of a neighborhood r, which can be constant or defined by a scalar field on the points: $r(P_i)$. We did not find a generally optimal solution for the weighting function, but implemented seven choices for our first investigations:

$$\omega_1 = 1 \qquad (2)$$

$$\omega_2 = 1 - x/r \qquad (3)$$

$$\omega_3 = 1 - (x/r)^2 \qquad (4)$$

$$\omega_4 = r/x^2 \qquad (5)$$

$$\omega_5 = \begin{cases} 1 - \frac{3}{2}a^2 + \frac{3}{4}a^3 & 0 \le a < 1 \\ \frac{1}{4}(2-a)^3 & 1 \le a < 2 \\ 0 & \text{otherwise} \end{cases} \qquad (6)$$

$$\omega_6 = \begin{cases} (\frac{5}{2} - b)^4 - 5(\frac{3}{2} - b)^3 + 10(\frac{1}{2} - v)^b & [0, \frac{1}{2}) \\ (\frac{5}{2} - b)^4 - 5(\frac{3}{2} - b)^3 & [\frac{1}{2}, \frac{3}{2}) \\ (\frac{5}{2} - b)^4 & [\frac{3}{2}, \frac{5}{2}) \\ 0 & [\frac{5}{2}, \infty) \end{cases} \qquad (7)$$

$$\omega_7 = \begin{cases} (3-c)^5 - 6(2-c)^5 + 15(1-c)^5 & [0,1) \\ (3-c)^5 - 6(2-c)^5 & [1,2) \\ (3-c)^5 & [2,3) \\ 0 & [3,\infty) \end{cases} \qquad (8)$$

with $a := \frac{2x}{r}$, $b := \frac{2.5x}{r}$ and $c := \frac{3.0x}{r}$, illustrated in Figure 1. The three functions ω_5, ω_6 and ω_7 are typical Gauss-like spline kernel functions used in smooth particle hydrodynamics (SPH) [10]. We use the same weighting functions for interpolating the vector field during Eigenvector integration. Even though, interpolation of Eigenvectors and interpolating tensors and locally computing its Eigenvectors lead to different results, we utilize the interpolation of the Eigenvector as a simpler implementation.

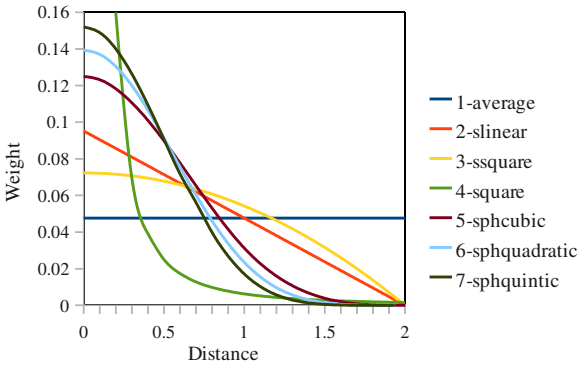


Figure 1: Different weighting functions of the distance interval 0.0 to 2.0, $r = 2.0$. Different slopes and characteristics are visualized. The *square* function (green) was clamped for axis scaling reasons and would grow further quadratically to the origin. The weights were normalized regarding to the integral of the curve in the interval. The curve numbers match the index of the weighting function: 1-average illustrates ω_1, 2-slinear illustrates ω_2, ...

We utilize tensor splats [1] for direct visualization of the tensor field. Figure 2 illustrates a point distribution along the edges of a rectangle and the corresponding tensor visualization with a neighborhood being $1/5$ of the longer rectangle edge. We then use Westin's shape

analysis method [24] to determine the so-called linear, planar and spherical shape factors. Points having a linearly distributed neighborhood are displayed as green oriented splats. Planar distributions are displayed as red disks. The linearity of the distribution tensor is shown in Figure 4 and Figure 5.

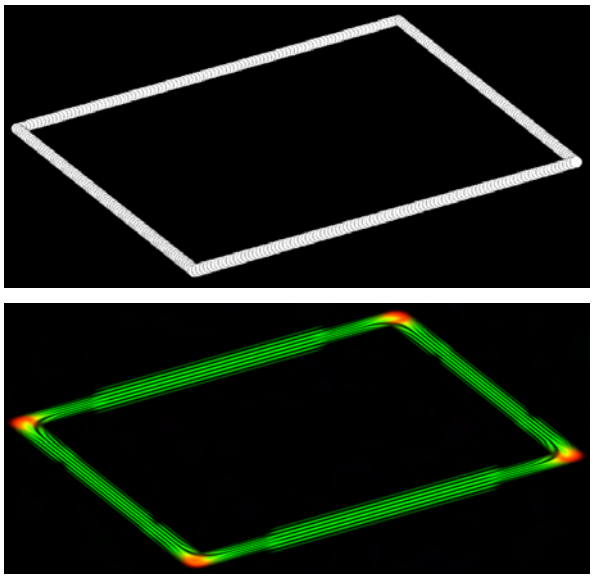


Figure 2: Distribution tensor visualization of a rectangular point distribution. *Top:* Points on a rectangle. *Bottom:* Tensor splats [1] of the point distribution tensor [21]. At each point one splat, a small textured and oriented disk, is drawn to represent the properties of the tensor's shape.

Visualizing streamlines is a common method to study vector fields. Starting from some seeding point, or initial condition, a curve $q(s)$ is computed which is always tangent to the vector field, solving the equation:

$$\dot{q}(s) = V(q(s)) \tag{9}$$

with s the curve parameter and V the vector field. Solving the differential equation at an arbitrary coordinate location Q within in the discretized data domain requires interpolation of the vector field. For mesh-free interpolation within a point cloud we use weighting functions parameterized with a specific radius of influence:

$$v(Q) = \frac{\sum_{i=1}^{N} v(P_i)\omega(|Q-P_i|,r)}{\sum_{i=1}^{N} \omega(|Q-P_i|,r)}, \tag{10}$$

with $v(P_i)$ representing the vector at point P_i.

2.2 Software Engineering Aspects

The algorithm was implemented using C++ within the VISH visualization shell [2]. The implementation extends a framework for computing integral geometries in vector fields, such as streamlines, pathlines or time surfaces. The streamline integration and visualization is separated into three different components: seeding, integration and displaying. The first component defines the initial conditions or seeding geometry. For computing streamlines within vector fields seeding points are sufficient. However, for streamlines of Eigenvector fields also an initial direction must be specified, because the Eigenvector is undirected. Integration based on an orientation continuing an user-chosen direction must be possible. Thus, requiring also a vector field on initial seeding points to disambiguate the Eigenvectors' orientations into unique directions.

Two new integration modules were developed. The first one extends the original streamline module, which was designed for vector field integration in uniform and curvilinear multi-block grids [4], to Eigenvector field integration. The second module expands this method further to allow integrating Eigenvector fields on mesh-free grids. One of the seven weighting functions (Equations 2, 3, 4, 5, 6, 7 and 8) and the radial influence weighting parameter can be specified for the interpolation of the Eigenvector inside the field domain. A range query on a KD-tree returns the points and their distances within the neighborhood of radius r. Equation 10 is utilized and Eigenvectors are aligned in orientation with respect to the Eigenvector of the closest neighbor. The Eigenvector is reversed if the dot product is negative. The integration of the streamline stops when the neighborhood becomes empty. Both integration modules support two different numeric schemes for the integration: explicit Euler and DOP853 [7]. Explicit Euler is used to get a fast yet inaccurate result. DOP853 is more expensive due to its adaptive stepsize but gives highly accurate results. When aiming at the same accuracy, DOP853 is faster than the Euler method by orders of magnitude. It is a Runge Kutta method of order eight using order five and three for error estimation and adaptive step size control, providing dense output. Accuracy measures and timing measures comparing the two integration methods were done, e.g., in [3].

The display module utilized here is reused from earlier development and implements color-coded illuminated lines utilizing OpenGL, allowing interactive navigation through the generated streamlines. Other modules, such as displaying ribbons [3] are also available.

2.3 Test Cases

We investigate the two Eigenvector integration modules on an uniform grid and on mesh-free grids. The Eigenvector field of a DTI-MRI scan [1], originally given on a uniform grid (128x128x56), was converted into a mesh-free grid, a point cloud holding the same Eigenvectors: Figure 3 (a) shows a volume rendering of the trace of the diffusion tensor along with the streamlines, revealing some brain structure and the location of the streamlines. Figure 3 (b) shows the comparison of Eigenvector streamlines computed on the uniform grid (blue) and Eigenvector streamlines computed in the point cloud (white). Both integrations were done with explicit Euler and a step size of 0.05. The size of a uniform grid cell is about 0.2, thus, utilizing about four integra-

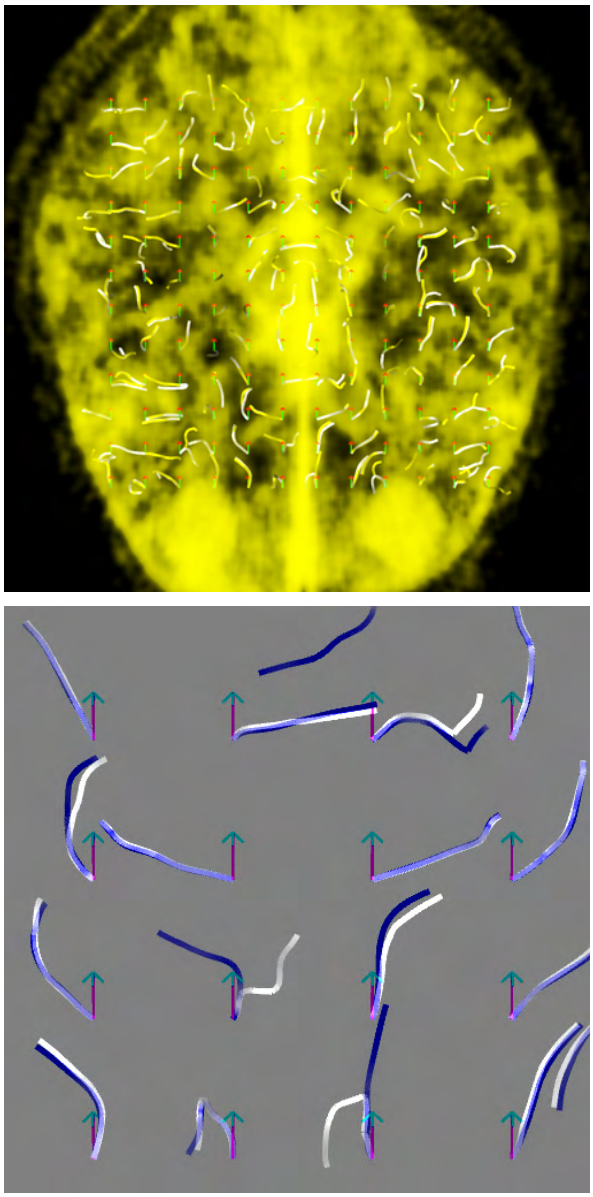
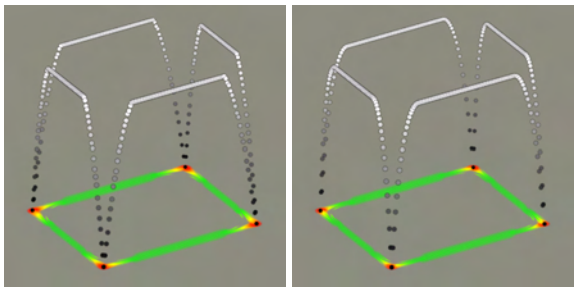
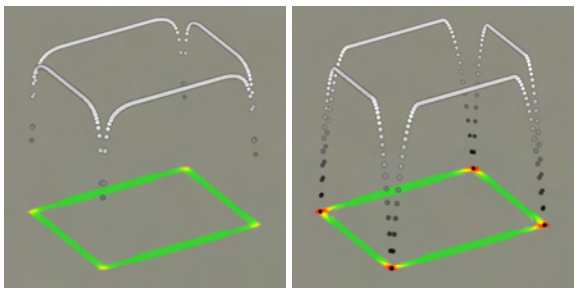


Figure 3: Comparison of the influence of integration of an Eigenvector field given on an uniform and a mesh-free grid. A mesh-free grid was generated from the uniform for testing. The arrows mark the start positions and directions of small Eigenstreamlines of a MRI diffusion tensor field. Streamlines on the uniform grid are blue. On the mesh-free grid they are white.

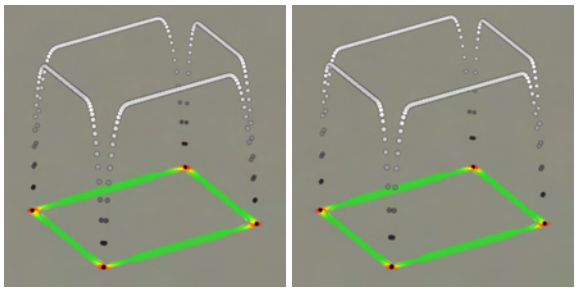
tion points per grid cell and requiring data interpolation within each cell. The length of each streamline is set to 1.0. Tri-linear interpolation was chosen for the uniform grid to compare the results with the linear weighting function ω_2 (*slinear*) for the mesh-free grid. The generated lines coincide on most cases. About 9% (13 of 144) do not coincide well. Some start in different directions. Here, the seeding vector field is almost perpendicular to the initial direction and the influence of the interpolation method results in different initial streamline directions. This issue could be cured by integrating Eigenvector streamlines in both directions starting from the initial seeding points, which would also allow avoiding the seeding vector field.



(a) average, slinear



(b) square, sphcubic

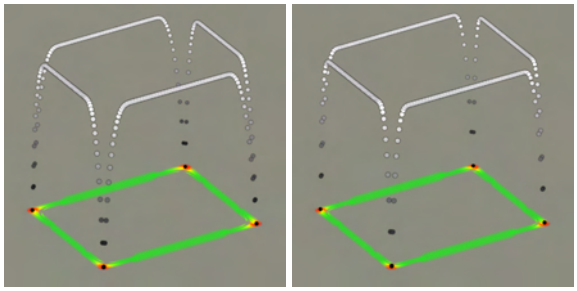


(c) sphquadratic, sphquintic

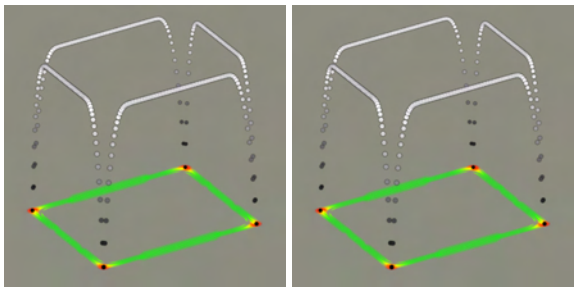
Figure 4: Influence of different weighting functions on the scalar field *linearity*, compare Figure 1. The linearity is illustrated by offset and over-scaling in z-axis, and gray-scale color-map on the points. Tensor splats directly show the distribution tensor.

Next, the influence of the different weighting functions on the computation of the distribution tensor was investigated. We define an analytic distribution of points along a rectangle as test case for computing the point distribution tensor. The rectangle is set up using a width of 10 and a height of 8. The radius parameter for the neighborhood is $r = 0.2$. Figure 4 illustrates the point distribution tensor using tensor splats and its corresponding linear shape factor by offsetting, over-scaling and a gray-scale color-map. The offsetting approach for the linear shape factor clearly illustrates the influence of the weighting: The "*average*" method resulting in a very abrupt change in the slope around corners points. The "*slinear*" weighting function results in smoother changes and a more localized influence, since closer points are weighted stronger than more distant points. *Square* shows the smoothest result. The three SPH spline kernels have an increasing locality with higher order of the kernel, when comparing *sphcubic*, *sphquadratic* and *sphquintic*. This is demonstrated in Figure 5 as well: Figure 5(a) shows the result of the

cubic and quadratic SPH kernel function. When the radius of the neighborhood is increased to match the kernels there is no visible difference between the *sphcubic* in Figure 5(a), *sphquadratic* and *sphquintic* in Figure 5(b) in the resulting *linearity*.



(a) *sphcubic* $r = 0.2$, *sphquadratic* $r = 0.2$



(b) *sphquadratic* $r = 0.25$, *sphquintic* $r = 0.3$

Figure 5: Different orders of the SPH kernel functions are compared, see Figure 1. *(a) sphcubic* and *shpquintic* using the same radius for the neighborhood. *(b) sphquadratic* and *shpquintic*, with adjusted neighborhood radius, have a similar result as the *sphcubic* (a)-left.

The influence of the integration scheme on the Eigenstreamline integration is demonstrated in Figure 6. The distribution tensor of a circular point distribution was computed using the *ssquare* weighting function. Tensor splats show the undirected Eigenvector, vector arrows show how the vector is directed within the internal vector representation. One Eigenstreamline is seeded downwards at the rightmost point of the circular point distribution and follows the undirected vectors. The top image shows Euler integration. Decreasing the step size would result in a more accurate integration. But, closing the gap of the integrated circle requires such a small step size, that the Runge Kutta method outperforms the Euler method. The 8^{th} order Runge Kutta method successfully closes the gap and reconstructs a circle from the circular point distribution, as shown in the bottom image. Also, a square-shaped point distribution was tested as shown in Figure 7. The length of a side is 10. Here, the influence of different weighting functions on the interpolation of the Eigenvector field was investigated. The distribution tensor was computed using the *ssquare* weighting function with $r = 2$. An Eigenstreamline is seeded downwards in the mid of the right edge. It follows the undirected vectors and flows around the corners of the rectangle. At each corner some error is introduced and the streamline is moving apart from

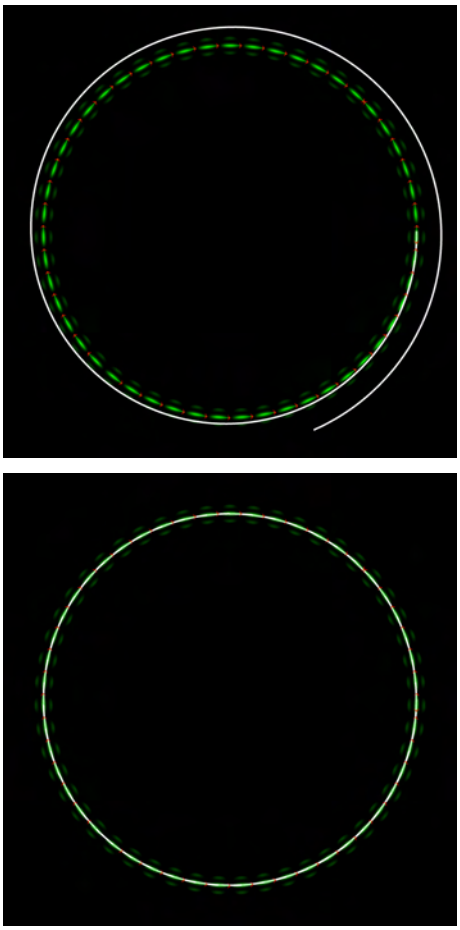


Figure 6: Comparison of different numerical integration schemes in a circular point distribution. One streamline (white) is seeded at the east-pole of the circle pointing southwards. Tensor splats and vector arrows illustrate the point distribution tensor and major Eigenvector. Note, that the Eigenvectors change orientation at north-east and south-west. *Top:* explicit Euler. *Bottom:* DOP853.

the original point distribution. Integration was done using the DOP853 method. Different weighting functions, mostly with $r = 1$, were tested for vector field interpolation. The length of the horizontal gap between the end and the start of the streamline was used as a measure for the integration error. Figure 8 shows the different errors in a bar diagram. The two best results were achieved using the *ssquare* and *average* weighting function.

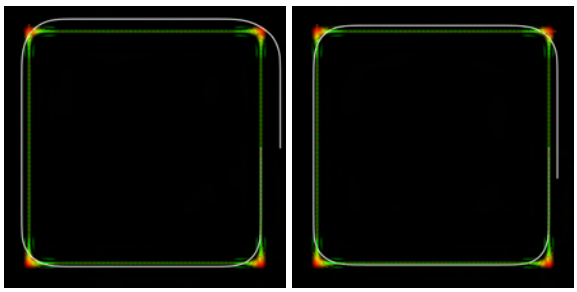


Figure 7: Comparison of Euler and DOP853 streamline integration on a square-shaped point distribution. Tensor splats and Eigenvectors are visualized besides the streamline (white) seeded downwards at the center of the right edge of the rectangle.

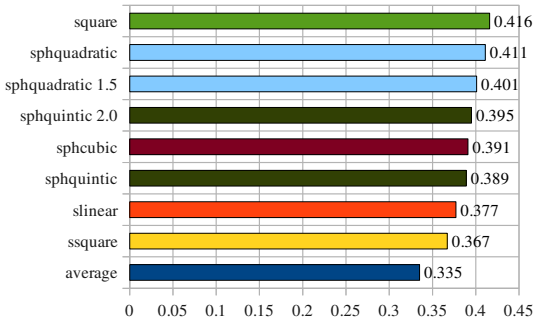


Figure 8: Comparison of errors in the square integration using different weighting functions for the vector interpolation. The weighting function for computing the tensor was *ssquare*, compare Figure 1. The values represent the horizontal distance between start and end point of the streamlines. The square's length is 10.0. The colors of the bars match the colors in Figure 1.

3 RESULTS

We used a dataset with circa eight million points covering a water basin close to the Danube in Austria. It was acquired by a Riegl's hydrographic laser scanner VQ-820G [22]. Figure 9 shows the point cloud colored by the *linearity* of a distribution tensor analysis. Here, we wanted to extract one power cable. The cable in the mid of the three lowest power cables suspended from the tall power pole. The white arrows mark the explicitly user-specified position and direction used as initial conditions of the streamline integration.

Different parameters and combinations of weighting functions for the tensor computation and the Eigenvector interpolation were investigated. The choice of a certain neighborhood radius and good weighting functions was crucial to successfully follow the 280 m long power cable. 41 parameter combinations were tested. For the tensor computation different radii $r = 0.5, r = 1.0$ and $r = 2.0$ and the weighting functions *average*, *slinear*, *ssquare* and the SPH kernels for the tensor were used. For the vector interpolation radii $r = 0.25, r = 0.5, r = 1.0, r = 2.0$ and $r = 3.0$ and all seven weighting functions were used. Figure 10 shows a view along the power cable, with a non optimal configuration. The Eigenstreamline is not following the cable to the end because it moves apart more than 1.0 m from the cable, resulting in an empty neighborhood during integration.

Best results were achieved by using the *ssquare* weighting with $r = 2.0$ for tensor computation and the *sphquintic* weighting with $r = 3.0$ for the vector interpolation. Results show that a more smooth weighting in the tensor computation and a more local interpolation weight are a good combination for reconstructing linear structures. Using the same weighting for tensor computation and vector interpolation did not work, see Figure 11 (b). The global error of the reconstruction at the end of the power cable is about 80 cm and needs to be further improved. The cable could only be followed using DOP853 integration. Explicit Euler failed to produce acceptable results. When comparing Figures $11(a)$ and $11(c)$ the global error is almost the same. The main difference is the local shape of the Eigenstreamline. A larger vector interpolation radius results in a smoother curve. Figure $11(c)$ shows the best reconstruction of the investigated technique and described parameters.

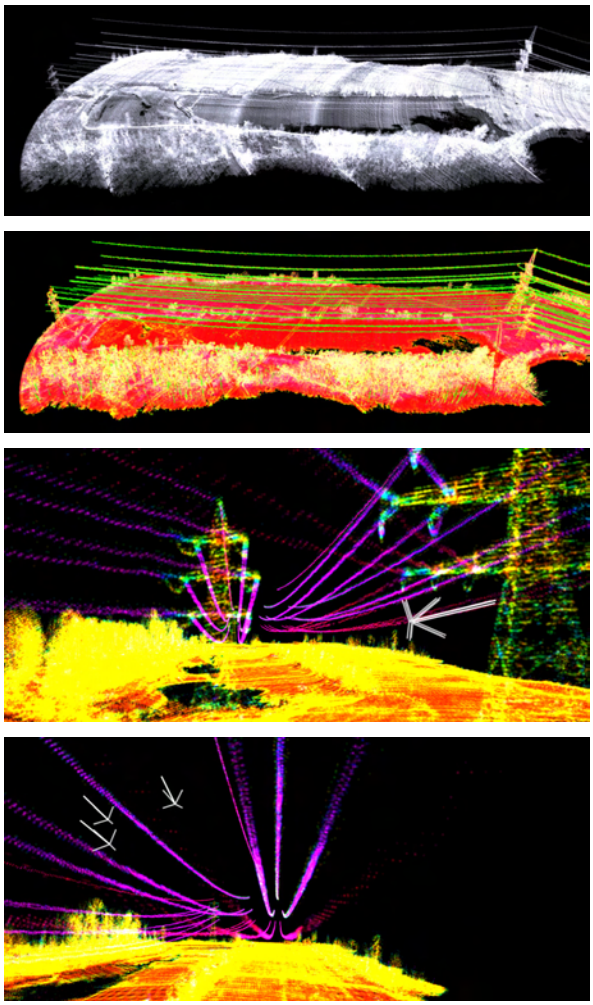


Figure 9: Overview of the LIDAR data set. The two upper images show the point cloud as points and as tensor splats (taken from [21]). In the two lower images points are colored by *linearity*. Three arrows mark the explicitly user-specified seeding points and directions of the streamline computation located at the mid lower power cable (magenta) of the larger power pole.

4 CONCLUSION

A new method of reconstructing power cables, or other linear structures in general, in point clouds was presented. The method employs the point distribution tensor as presented in previous work [21]. Different weighting functions for the tensor computation and the interpolation of the major Eigenvector field were implemented and compared. Streamline integration was verified on artificial test cases and applied to a LIDAR point cloud dataset acquired from actual observations. Finally, a power cable was reconstructed and visualized using this dataset.

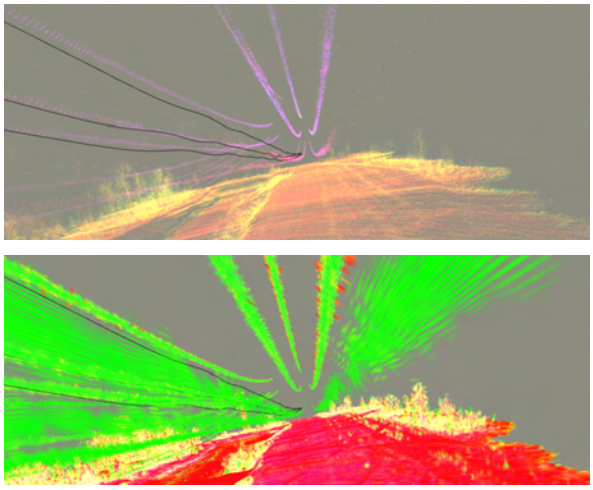
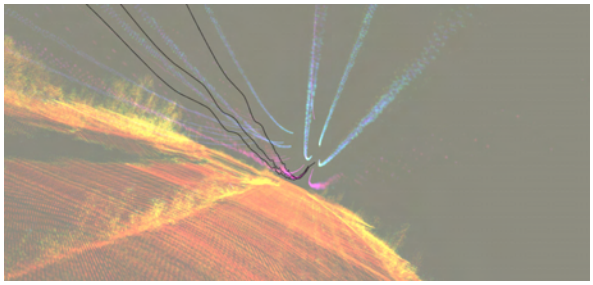
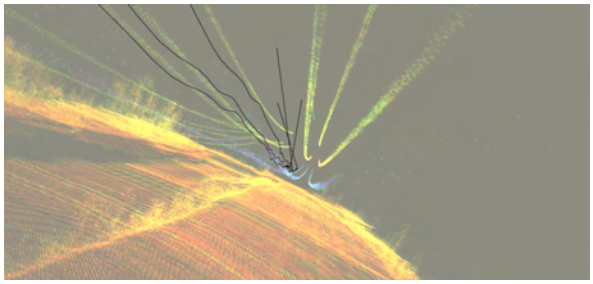


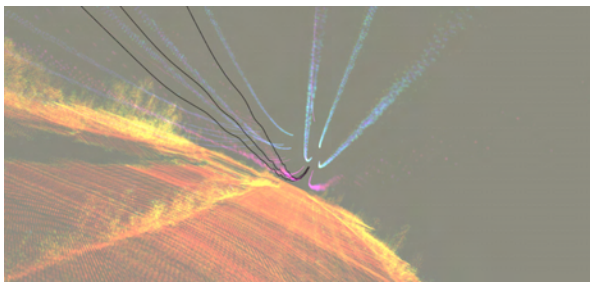
Figure 10: Power cable reconstruction via streamlines. The distribution tensor was computed using the *average* $r = 2.0$ weighting and the vector interpolation was done with the *ssquare* $r = 1.0$ weighting. *Top:* Points colored by *linearity*. *Bottom:* Tensor splats illustrate the distribution tensor. Streamlines are moving apart from the power cable and break before they can reconstruct the full 280 m of cable.



(a) Tensor: *ssquare r=2*, Vectorfield: *sphquintic r=1*



(b) Tensor: *sphquintic r=2*, Vectorfield: *sphquintic r=2*



(c) Tensor: *ssquare r=2*, Vectorfield: *sphquintic r=3*

Figure 11: Comparison of different parameters and weighting function combinations of the computation, finally resulted in a successfully reconstructed power cable. The LIDAR point cloud is colored by *linearity* of the distribution tensor. The three Eigenvector streamlines reconstruct a 280 m long cable.

5 FUTURE WORK

Other weighting functions for computing the tensor and doing the interpolation during the streamline integration need to be tested. Automatic determination of the optimal combination of weighting functions and also their parameters will be the goal of further investigations. Seeding points and directions for computing the streamlines need also to be chosen automatically, for example, by taking tensor properties into account. Following the major Eigenvector of points with high *planarity* or *sphericity* needs to be prevented during streamline integration. Finally, more datasets should be explored to stabilize the method. Furthermore, minor changes of the algorithm would enable streamline integration in datasets stemming from SPH simulations.

ACKNOWLEDGMENT

Many thanks to Frank Steinbacher for proving the LIDAR data. This work was supported by the Austrian Science Foundation FWF DK+ project *Computational Interdisciplinary Modeling* (W1227) and grant P19300. This research employed resources of the Center for Computation and Technology at Louisiana State University, which is supported by funding from the Louisiana legislature's Information Technology Initiative. This work was supported by the Austrian Ministry of Science BMWF as part of the UniInfrastrukturprogramm of the Forschungsplattform Scientific Computing at LFU Innsbruck.

REFERENCES

[1] W. Benger, H. Bartsch, H.-C. Hege, H. Kitzler, A. Shumilina, and A. Werner. Visualizing Neuronal Structures in the Human Brain via Diffusion Tensor MRI. *International Journal of Neuroscience*, 116(4):pp. 461–514, 2006.

[2] W. Benger, G. Ritter, and R. Heinzl. The Concepts of VISH. In 4^{th} *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007.

[3] W. Benger and M. Ritter. Using geometric algebra for visualizing integral curves. In E. M. S. Hitzer and V. Skala, editors, *GraVisMa 2010 - Computer Graphics, Vision and Mathematics for Scientific Computing*. Union Agency - Science Press, 2010.

[4] W. Benger, M. Ritter, S. Acharya, S. Roy, and F. Jijao. Fiberbundle-based visualization of a stir tank fluid. In 17^{th} *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 117–124, 2009.

[5] T. E. Conturo, N. F. Lori, T. S. Cull, E. Akbudak, A. Z. Snyder, J. S. Shimony, R. C. Mckinstry, H. Burton, and M. E. Raichle. Tracking neuronal

fiber pathways in the living human brain. *Proc Natl Acad Sci U S A*, 96(18):10422–10427, Aug. 1999.

[6] M. Descoteaux, L. Collins, and K. Siddiqi. A multi-scale geometric flow for segmenting vasculature in mri. In *In Medical Imaging Computing and Computer-Assisted Intervention*, pages 500–507, 2004.

[7] S. N. E. Hairer and G. Wanner. *Solving ordinary differential equations I, nonstiff problems, 2nd edition*. Springer Series in Computational Mathematics, Springer-Verlag, 1993.

[8] L. A. Fernandes and M. M. Oliveira. Real-time line detection through an improved hough transform voting scheme. *Pattern Recognition*, 41(1):299 – 314, 2008.

[9] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32 – 40, jan 1975.

[10] D. A. Fulk and D. W. Quinn. An analysis of 1D smoothed particle hydrodynamics kernels. *Journal of Computational Physics*, 126(1):165–180, 1996.

[11] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2001.

[12] D. K. Jones. Studying connections in the living human brain with diffusion mri. *Cortex*, 44(8):936 – 952, 2008.

[13] Y. Jwa, G. Sohn, and H. B. Kim. Automatic 3d powerline reconstruction using airborne lidar data. *IAPRS*, XXXVIII(2004):105–110, 2009.

[14] K. Kraus and N. Pfeifer. Determination of terrain models in wooded areas with airborne laser scanner data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 53(4):193 – 203, 1998.

[15] Z. Li, R. Walker, R. Hayward, and L. Mejias. Advances in vegetation management for power line corridor monitoring using aerial remote sensing techniques. In *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, pages 1 –6, oct. 2010.

[16] Y. Liu, Z. Li, R. F. Hayward, R. A. Walker, and H. Jin. Classification of airborne lidar intensity data using statistical analysis and hough transform with application to power line corridors. In *Digital Image Computing : Techniques and Applications Conference (DICTA 2009)*, Melbourne, Victoria, December 2009. IEEE Computer Society.

[17] T. Melzer. Non-parametric segmentation of als point clouds using mean shift. *Journal of Applied Geodesy*, 1(3):159–170, 2007.

[18] T. Melzer and C. Briese. Extraction and modeling of power lines from als point clouds. In *Proceedings of 28th Workshop*, pages 47–54. Österreichische Computer Gesellschaft, 2004. talk: Austrian Association for Pattern Recognition (ÖAGM), Hagenberg; 2004-06-17 – 2004-06-18.

[19] S. Mills, M. Gerardo, Z. Li, J. Cai, R. F. Hayward, L. Mejias, and R. A. Walker. Evaluation of aerial remote sensing techniques for vegetation management in power line corridors. *IEEE Transactions on Geoscience and Remote Sensing*, October 2009.

[20] M. Persson, J. Solem, K. Markenroth, J. Svensson, and A. Heyden. Phase contrast mri segmentation using velocity and intensity. In R. Kimmel, N. Sochen, and J. Weickert, editors, *Scale Space and PDE Methods in Computer Vision*, volume 3459 of *Lecture Notes in Computer Science*, pages 119–130. Springer Berlin - Heidelberg, 2005.

[21] M. Ritter, W. Benger, B. Cosenza, K. Pullman, H. Moritsch, and W. Leimer. Visual data mining using the point distribution tensor. In *IARIS Workshop on Computer Vision and Computer Graphics - VisGra 2012*, Feb-Mar 2012.

[22] F. Steinbacher, M. Pfennigbauer, A. Ulrich, and M. Aufleger. Vermessung der Gewassersohle - aus der Luft - durch das Wasser. In *Wasserbau in Bewegung ... von der Statik zur Dynamik. Beitrage zum 15. Gemeinschaftssymposium der Wasserbau Institute TU München, TU Graz und ETH Zürich*, 2010.

[23] D. Weinstein, G. Kindlmann, and E. Lundberg. Tensorlines: advection-diffusion based propagation through diffusion tensor fields. In *Proceedings of the conference on Visualization '99: celebrating ten years*, VIS '99, pages 249–253, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[24] C. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, and F. Jolesz. Geometrical diffusion measures for mri from tensor basis analysis. In *Proceedings of ISMRM, Fifth Meeting, Vancouver, Canada*, page 1742, Apr. 1997.

Real-time Mesh Extraction of Dynamic Volume Data Using GPU

Szymon Engel
AGH, Krakow, Poland
szymon@hoopoe.com.pl

Witold Alda
AGH, Krakow, Poland
alda@agh.edu.pl

Krzysztof Boryczko
AGH, Krakow, Poland
boryczko@agh.edu.pl

ABSTRACT

In the paper an algorithm of triangle mesh generation for a full three-dimensional volumetric data is presented. Calculations are performed in real time using graphics processors. The method is very well suited for the visualization of dynamic data, as the calculations use only the current frame data (not including data from previous frames). Due to high performance of the algorithm, it can be practically applied in programs for digital sculpting, simulators and games which require editable geometries.

Keywords

mesh generation, iso-surface, volume data, real-time, GPU computation

1 INTRODUCTION

Nowadays, visualization of volumetric data is very often applied in practice. Both in medicine, e.g. for the MRI, PET, or CT data presentation in medical imaging, nondestructive inspection of materials (industrial CT), digital sculpting software, as well as in computer games. Often the visualization itself is not sufficient and a three-dimensional mesh is required for a physical calculations, such as collision detection, calculation of material properties or stress. Moreover, the advantage of representing models with triangle meshes is that modern GPUs are optimized for efficient rendering of triangles.

Another issue is that volumetric data can change dynamically. When modeling is performed in a program for sculpting a virtual material, a three-dimensional mesh generated in real time is needed in order to display the results. In video games or simulators of earth-moving machineries, we have to deal with the terrain, which cannot be fully represented by height maps. We may require a visualization of structures such as tunnels, caves, overhangs of the land as well as other modifications caused as a result of a player actions, such as explosions, vehicle interactions with the terrain or other gameplay activities.

Currently available methods often do not allow to generate a mesh for a large amount of data in real time. Sometimes the resulting effect is described as "interactive" which usually means ability to carry out calculations giving a few frames per second. This speed, however, is not sufficient for the smooth operation of applications such as computer games.

In response to these problems, a mesh generation algorithm for a fully three-dimensional volumetric data has been developed. All calculations are performed on GPU in real time by which we understand the actual mean speed of computations above 30 frames per second. Another advantage of the presented algorithm is its independence of the volumetric data representation; therefore, scalar fields, implicit functions, or metaball objects can be used.

The rest of this article is organized as follows. The next section presents previous work on mesh generation methods of volumetric data. The third part briefly describes the possible methods of data representation. It then shows the subsequent steps of the algorithm while in the following passage detailed information on how to implement the algorithm using graphics processors is included. The last section presents a description and summary of results.

2 RELATED WORK

The subject of this paper is to visualize an iso-surface generated for the volumetric data using triangle mesh. Mesh-based methods allow their easy integration with other algorithms or modules, e.g. physics engines, that require mesh as an input. Thanks to this approach our algorithm can be used in real-time applications.

Therefore, discussion of previous works does not include ray-tracing and view-dependent visualization; however, information on the methods of these types can be found e.g. in [OBA05, LK10, GDL+02].

GPU-based methods which allows to interactive or real-time calculations were presented e.g. in [KOKK06, CNLE09, KOR08, TSD07].

One of the basic methods of generating a mesh on the basis of volumetric data is the Marching Cubes algorithm [LC87] developed in the 1980s. It consists in dividing the visualized space into equal cubes and generating a polygon within each cube, which is then subjected to triangulation. The position and shape of the polygon are dependent on the values of an implicit function in eight vertices of each cube. A common issue with this method is that it requires generating a mesh in the entire space of visualized data. In response to this, several hierarchical and adaptive versions of the Marching Cubes algorithm have been developed [OR97, Blo88, WKE99, KKDH07] using octal trees to reduce the area for which calculations were carried out and which reduce the number of triangles by generating them in different densities depending on the distance of the camera. Implementations of of the Marching Cubes algorithm using GPUs are presented in [JC06, Gei07].

Another, possibly less popular, method is the SurfaceNets method [Gib98] which originally was used to visualize binary medical data. This method is dual to the marching cubes algorithm and, as in the latter one, visualized space is divided into cubes of the same dimensions. In its base version, the method consisted of selecting the nodes belonging to the surface in a way that a certain node has been selected, if among the eight vertices of the cube were those that had a different sign. These nodes were linked with adjacent ones thus creating a net, which was then smoothed by moving the nodes so as to minimize the energy between them while maintaining the restriction that a node could not leave the cube, to which it originally belonged. The final step was a triangulation of the network, which gave the resulting mesh of triangles. The next method which belongs to the group of "dual algorithms" is the one by [Nie04], which generates a mesh very similar to SurfaceNets, but its implementation is more like the Marching Cubes algorithm. One of the important differences between this method and the SurfaceNets is that the mesh generated by the former is a proper two-dimensional manifold.

In addition to the methods outlined above, there are also: the marching tetrahedra method [TPG99], whose operating principle is based on the marching cubes algorithm, the marching triangles method based on the Delaunay triangulation [HSIW96] which generates an irregular mesh, or the method based on marching cubes which allows one to obtain sharp edges, described in the work [JLSW02].

We follow the existing approach of dual marching cubes, however, our algorithm is implemented exclusively on GPU and it efficiently exploits geometry shaders. Thanks to the use of dual methods, the resulting mesh contains fewer triangles and is regular due to the number of generated triangles within each of the cubes. The latter allows the method to be implemented in a very efficient way using graphics processors. Former GPU-accelerated mesh extraction algorithms (e.g. [KW05], [Goe05], [JC06]) are based on both CPU and GPU computations, using vertex or fragment shaders only. Although meshes generated using our method are not proper two-dimensional manifolds, our approach is extremely efficient and can be used for dynamic data which change on random every frame.

3 VOLUMETRIC DATA

The basic method of describing three-dimensional volumetric data is the implicit function. By setting the values of the contour we obtain surface limiting the desired area. If the values of this function represent the Euclidean distances from a given contour and we save them as an array, then we get a three dimensional signed distance field $D : \mathbb{R}^3 \to \mathbb{R}$, representing the iso-surface S, defined for point $p \in \mathbb{R}^3$ as:

$$D(p) = sgn(p) \cdot \min\{|p-q| : q \in S\}, \qquad (1)$$

where

$$sgn(p) = \begin{cases} -1 & \text{if p is inside} \\ +1 & \text{if p is outside} \end{cases} \qquad (2)$$

This representation can be stored in graphics card memory as a three-dimensional texture. Thanks to this representation, smoothing of the resulting mesh using normals computed directly from distance fields and vertex distances from the surface, is very effective.

Also, most medical data is stored in the form of three-dimensional arrays. For such data combined with contour values we can generate a mesh. Another, less common way to represent the volumetric data, is using metaball objects which, with adequate representation, can be converted to distance fields.

4 ALGORITHM OVERVIEW

Due to the GPU architecture and the way they carry out calculations, the developed algorithm is based on dual methods. They allow to obtain a regular mesh consisting of squares, so one doesn't need expensive triangulation of polygons generated inside a cube, as is the case of marching cubes method. The algorithm has been adapted to carry out calculations on GPUs, and highly parallelized, which allows it to achieve very high performance.

Input data block, of size n^3, where n of a form 2^k, is divided into equal cubes. This is shown in Figure 1.

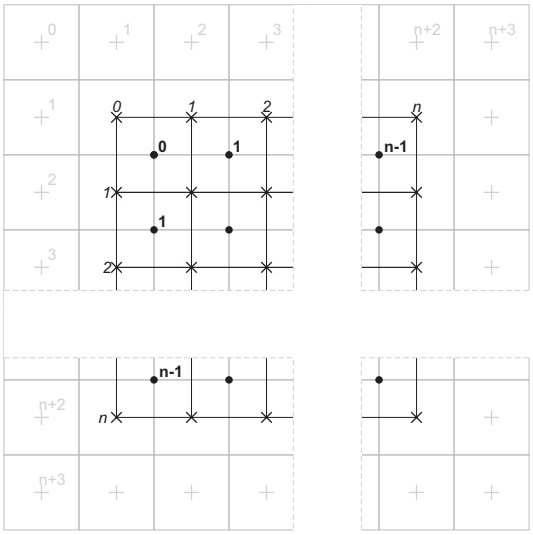


Figure 1: Data organization (2D view)

The image represents a two-dimensional version, but the three-dimensional one is similar.

Calculation points marked with "•" symbol are placed in the centers of cubes and their coordinates are used to calculate the distance from the contour, based on the volumetric data. Distance field values are stored in points marked "+", while points marked "×" represent vertices of quadrangles generated by the algorithm.

The calculations of the algorithm are processed in two steps. At the beginning, for each calculation point $p \in \mathbb{R}^3$, marked with the symbol "•", such that $p \in \{[0,n-1] \times [0,n-1] \times [0,n-1]\}$, there are generated three points $q,r,s \in \mathbb{R}^3$ such that for $p = (x,y,z)$: $q = (x-1,y,z)$, $r = (x,y-1,z)$, $s = (x,y,z-1)$. In each of these points p,q,r,s an implicit function value $d_p = f(p)$ is calculated and three edges defined by pairs of calculation points pq, pr, ps are created. Then, for each edge – if its endpoints have different signs (lie on different sides of the contour) – a quadrangle located on the border of the cubes is generated. Its orientation is determined by the direction of the edge, which is consistent with the direction of the normal to the surface of the quadrangle. A set of squares, generated in this way, approximates the iso-surface for input volumetric data, and is smoothed in the next stage. As a result of conversion of each square to a pair of triangles, a triangle mesh is obtained.

Due to the fact that during the calculation the sign changes, zero is treated differently depending on the direction of the edge, the condition 3, under which quadrilaterals are generated is presented as follows:

$$(f(q) \geq 0 \wedge f(p) < 0) \vee (f(q) < 0 \wedge f(p) \geq 0) \quad (3)$$

The first step in the algorithm, described above, is ideally suited for parallelization, because the calculations for computing the individual points can be carried out independently. Despite the fact that the distance fields are calculated for each point twice, it is possible to obtain a high-performance computing algorithm, because it is not a costly operation.

The second stage of the algorithm is a smoothing of generated mesh by moving its vertices in the direction of the surface represented by the distance fields values. For this purpose, at each vertex of the distance field, a normal vector n is calculated on the basis of the gradient and the distance d to the surface. Then, the vertex is moved in the direction of the normal, by the value calculated according to formula 4.

$$p' = p + dn \quad (4)$$

In the case that the resulting mesh is used only for displaying a surface, this step can be implemented directly during rendering. Otherwise, it is possible to smooth the mesh only and use it for subsequent calculations, such as collision detection.

The advantage of the developed algorithm over marching cubes method consists partly in that for creating a quad for the cube we generate exactly four indices and there is no need for triangulation of polygons generated. This allows the calculations to be successfully performed on the GPU.

5 IMPLEMENTATION DETAILS

The algorithm was implemented using the OpenGL graphics library; however, DirectX library can be used as well.

Input volumetric data, for which a mesh is generated, is divided into equal blocks, for which fragments of the mesh are generated independently. This approach was chosen because of the GPU hardware limitations on the maximum size of supported textures, as well as for optimization purposes. To be specific: not all parts of dynamic data need to be modified at the same time, and hence there is no need for mesh regeneration in these blocks. Generated meshes merge together in continuous blocks along borders.

The algorithm is carried out equally for each data block. All calculations are done on the GPU, so the data are not transferred between main memory and graphics card memory. Calculations are carried out in two phases, the algorithm flowchart is shown in Figure 2.

In both passes of the rendering shaders refer to the volumetric data.

5.1 Volume Representation

Three-dimensional floating-point texture is used for distance field representation in each block of data.

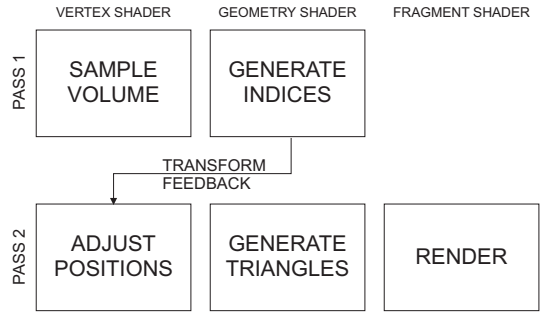


Figure 2: Algorithm flowchart

Texture size is $(n+3)^3$, 2 bytes per texel. Although 4-byte floating point numbers can be applied for the precision improvement, no need for that has been found during testing. To avoid communication between neighboring blocks, block's data size is enlarged and overlaps adjacent blocks; for each common quadrangle between two adjacent blocks of size $(n+3)^3$ each, $2n^2$ of voxels are duplicated. This allows parts of the mesh in each block to be generated independently of the others.

In the texture, we used a single GL_RED channel, where the data is stored in GL_FLOAT16 format. In order to allow reading distance field values, each point of the texture is sampled using linear filtering. This is done automatically, and the values are calculated on the basis of neighboring texels. Mipmaps for the textures are not generated.

5.2 Mesh Generation (Pass 1)

In the first pass calculations are performed using vertex shader and geometry shader. Then, using a transform feedback, generated indices are stored in the output buffer. Input and output data are presented in Table 1.

IN
Vertex buffer containing calculation points $p \in \{[0,n-1]\times[0,n-1]\}$ *type:* GL_ARRAY_BUFFER *size:* $(n-1)^2$ *sharing:* one per all blocks
Texture with volumetric data *type:* GL_TEXTURE_3D *size:* $(n+3)^3$ *sharing:* one per single block
OUT
Buffer with indices of generated quads *type:* GL_ELEMENT_ARRAY_BUFFER *sharing:* one per single block

Table 1: Input and output data of the first pass

The subsequent steps of the algorithm are as follows:

1. Data from the vertex buffer containing the calculation points is rendered $n-1$ times, using the instance rendering.

2. For a calculation point p three points q,r,s are created in the vertex shader.

3. For each of the points p,q,r,s a volumetric texture is sampled in order to retrieve implicit function value. Texture coordinates t are calculated according to formula 5.

$$t = \frac{p+2}{n+3} \tag{5}$$

4. Subsequently three edges pq, pr, ps are created and according to formula 3 it is checked whether the values of implicit function at the endpoints have different signs.

5. For each edge, if the sign changes, there is a flag set, which specifies whether the quadrilateral is generated or not, and what is its orientation (zero means that the quadrilateral is not generated). Generated flags along with the coordinates of the point p are forwarded to the geometry shader.

6. In the geometry shader, for each non-zero flag there a vertex containing four indices of generated quadrangles is established. Indices are calculated on the basis of the flag f and coordinates p. For example, quadrangle, which normal is consistent with the direction of the edge pq, is defined by (i_1,i_2,i_4,i_3), where

$$\begin{aligned} i_1 &= p_x n^2 + p_y n + p_{z+1} \\ i_2 &= p_x n^2 + p_{y+1} n + p_{z+1} \\ i_3 &= p_x n^2 + p_{y+1} n + p_z \\ i_4 &= p_x n^2 + p_y n + p_z \end{aligned}$$

7. Using the feedback transformation these indices are stored directly in the index buffer, used in the next pass. Number of saved indices is queried using an OpenGL query mechanism.

5.3 Rendering (pass 2)

The second pass is responsible for smoothing of the generated mesh and its rendering. All programmable shader units, i.e. vertex, fragment and geometry shaders are used. The input data is presented in Table 2.

Subsequent steps of the second pass of the algorithm are as follows:

1. The data from the vertex buffer is rendered using the indices as primitives of the GL_LINES_ADJACENCY type. This type was chosen because it is the only type that can render

IN
Vertex buffer which contains all potential vertices, such as: $u \in \{[0,n] \times [0,n] \times [0,n]\}$. *type:* GL_ARRAY_BUFFER *size:* n^3 *sharing:* one per all blocks
Buffer with indices for generated quadrangles *type:* GL_ELEMENT_ARRAY_BUFFER *sharing:* one per single block
Texture with volumetric data *type:* GL_TEXTURE_3D *size:* $(n+3)^3$ *sharing:* one per single block

Table 2: Input data for the second pass of the algorithm

primitives indexed by four indices (in OpenGL version 3.0 or higher it is not possible to render quadrangles).

2. Then in the vertex shader for each vertex u, a normal vector n is calculated, on the basis of the gradient map. Normal value is obtained by sampling the volumetric data texture in the neighboring six texels in x, y, z directions.

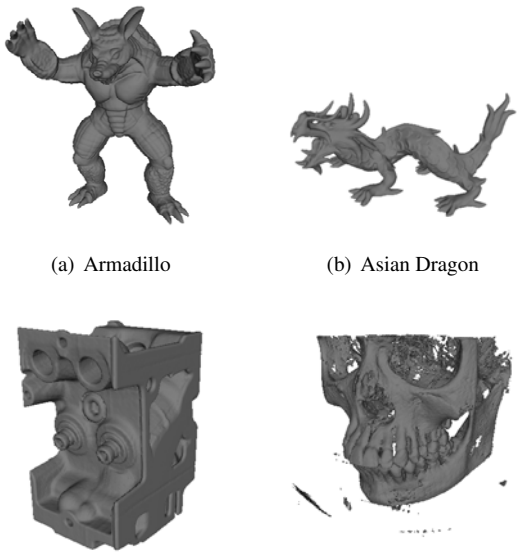
3. On the basis of the direction of the normal and the density function value, the point u is moved in the direction of the contour, according to formula 4. Due to the fact that the value of the density function is calculated as the average of neighboring texels at the point u, for small values of n it is required to perform a smoothing of the mesh in an iterative manner.

4. Vertices calculated in this way are sent to the geometry shader, in which there is a change of type done, from "lines adjacency" into "triangle strip".

5. The last step is to display a completed mesh, during which the associated shading calculations are performed in the fragment shader. In case when mesh rendering is not required, but the mesh is needed for further calculations, smoothed values of the vertices can be stored in the output buffer using a transform feedback.

6 RESULTS

All calculations were performed on an AMD Phenom II X6 1090T 3.2GHz computer with an nVidia GeForce GTX 460 graphics card. Figure 3 presents datasets used in tests; Armadillo and Asian Dragon from *The Stanford 3D Scanning Repository*, Engine and Skull from `http://www.volvis.org/`. In addition, three-dimensional animated Perlin noise has been used as a dynamic, time-dependent data. All tests were run for different n on one block of data.



(a) Armadillo (b) Asian Dragon

(c) Engine (d) Skull

Figure 3: Datasets used for tests

Table 3 lists times for mesh generation and rendering for different block sizes and different data sets. All meshes were regenerated from volumetric data each frame; moreover, calculations for time-dependent Perlin noise data were also performed before mesh generation every frame. Generated meshes for the Armadillo dataset for different block sizes are presented on figure 4, results for noise dataset are presented on figure 6.

dataset	n	triangle count	fps
Armadillo	64	14180	1100
Armadillo	128	91958	208
Armadillo	256	494880	28
Asian Dragon	64	5864	1180
Asian Dragon	128	41578	234
Asian Dragon	256	229840	30
Engine	256	592884	29
Skull	256	1699526	23.6
Perlin Noise	128	180k-246k	60

Table 3: Results for different block sizes and data sets

Table 4 presents results for our method compared to [Goe05, JC06]. The Engine and Skull datasets were used, no preprocessing were performed for these data. As it can be seen our algorithm performs much faster, however, if all methods would be run on the same hardware configuration, the difference could be less significant. Both methods of [Goe05] and [JC06] were tested on nVidia GeForce 6800GT graphics card.

The Marching Cubes algorithm described in [Gei07] seems to execute faster than [Goe05, JC06] methods but no measurable results were published. Authors of [Gei07] claims that their algorithm executes in interac-

(a) $n = 64$ (b) $n = 128$ (c) $n = 256$

Figure 4: Generated mesh for the Armadillo dataset

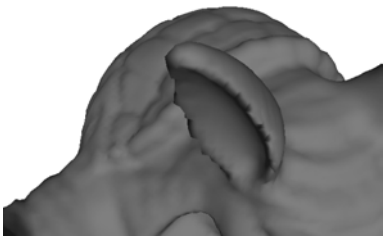
tive frames but mesh generation are not performed each frame.

dataset	size	method	fps
engine	256x256x110	[Goe05]	3.6
engine	256x256x128	[JC06]	2.8
engine	256x256x256	our method	29
skull	256x256x225	[Goe05]	2.4
skull	256x256x256	[JC06]	1.5
skull	256x256x256	our method	23.6

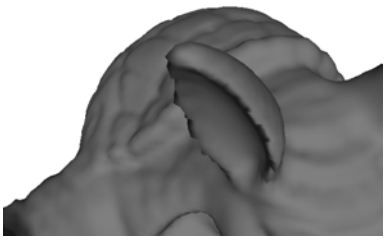
Table 4: Results compared to previous GPU-based methods

7 MESH IMPROVEMENTS

The presented method works well for smooth surfaces, such as the ones presented in figure 6. In case of surfaces with sharp edges we see artifacts as it is shown in figure 5(a).



(a) Artifacts on sharp edges



(b) Smoothed mesh

Figure 5: Mesh improvements due to smoothing

In order to improve visual quality of the generated surface, in the second pass of the algorithm a smoothing process based on normals is performed about 10 times. Next, during transforming quadrangles into strips of triangles, the quadrangles are divided along the shorter diagonal. The last step of smoothing the mesh is computation of normals for every pixel in fragment shader, on the basis of volumetric data. Normals are calculated in the same way as for the mesh vertices. The smoothed mesh is presented in Figure 5(b).

8 CONCLUSION AND FUTURE WORK

In this paper we present a real-time algorithm for generating a three-dimensional mesh fully based on volumetric data. This method has been optimized for graphics processors and provides a significant advantage over the already existing solutions for conventional processors. The presented algorithm is also very well suited for the visualization of dynamic data, because the calculations carried out do not need to know the state of the algorithm from previous time steps.

With the resulting performance, practical application of the algorithm in digital sculpting software, earth-moving machineries simulators and computer games is fully possible. The tests show a significant advantage of GPUs. The volumetric data representation that has been used allows also for efficient data modification using GPUs.

As part of further work on the algorithm it would be reasonable to add support for levels of detail (LOD), so as to enable the process to connect continuously adjacent blocks containing cubes of different sizes and densities.

The second issue is to optimize the algorithm by an additional parallelization and simultaneous calculation carried out for 4 blocks. It would be possible in the case of using all four available texture channels. As a result, it would be possible to generate meshes for the four blocks at the same time.

9 ACKNOWLEDGEMENTS

The work described in this paper was partially supported by The European Union by means of European

Social Fund, PO KL Priority IV: Higher Education and Research, Activity 4.1: Improvement and Development of Didactic Potential of the University and Increasing Number of Students of the Faculties Crucial for the National Economy Based on Knowledge, Subactivity 4.1.1: Improvement of the Didactic Potential of the AGH University of Science and Technology "Human Assets", No. UDA-POKL.04.01.01-00-367/08-00.

One of us (WA) kindly acknowledges partial support by Ministry of Science and Higher Education in Poland, project No. N N519 443039.

10 REFERENCES

[Blo88] J. Bloomenthal. Polygonization of implicit surfaces. *Comput. Aided Geom. Des.*, 5(4):341–355, 1988.

[CNLE09] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: ray-guided streaming for efficient and detailed voxel rendering. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 15–22, New York, NY, USA, 2009. ACM.

[GDL+02] Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. Interactive view-dependent rendering of large isosurfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 475–484, Washington, DC, USA, 2002. IEEE Computer Society.

[Gei07] Ryan Geiss. *Generating Complex Procedural Terrains using the GPU*, chapter 1. GPU Gems. Addison Wesley, 2007.

[Gib98] Sarah F. Frisken Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *MICCAI '98: Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 888–898, London, UK, 1998. Springer-Verlag.

[Goe05] F. et al. Goetz. Real-time marching cubes on the vertex shader. In *In Proceedings of Eurographics 2005, Dublin, Ireland , pp. 5-8*, 2005.

[HSIW96] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Marching triangles: range image fusion for complex object modelling. In *Proc. Conf. Int Image Processing*, volume 1, pages 381–384, 1996.

[JC06] Gunnar Johansson and Hamish Carr. Accelerating marching cubes with graphics hardware. In *In CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, ACM*, page 378. Press, 2006.

[JLSW02] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, New York, NY, USA, 2002. ACM.

[KKDH07] Michael Kazhdan, Allison Klein, Ketan Dalal, and Hugues Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 125–133, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[KOKK06] Takashi Kanai, Yutaka Ohtake, Hiroaki Kawata, and Kiwamu Kase. Gpu-based rendering of sparse low-degree implicit surfaces. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, GRAPHITE '06, pages 165–171, New York, NY, USA, 2006. ACM.

[KOR08] John Kloetzli, Marc Olano, and Penny Rheingans. Interactive volume isosurface rendering using bt volumes. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 45–52, New York, NY, USA, 2008. ACM.

[KW05] P. Kipfer and R. Westermann. Gpu construction and transparent rendering of isosurfaces. In *Vision, Modeling, and Visualization (VMV 2005), Erlangen, Germany, Nov. 16-18*, 2005.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[LK10] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 55–63, New York, NY, USA, 2010. ACM.

[Nie04] Gregory M. Nielson. Dual marching cubes. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 489–496, Washington, DC, USA, 2004. IEEE Computer Society.

[OBA05] Yutaka Ohtake, Alexander Belyaev, and

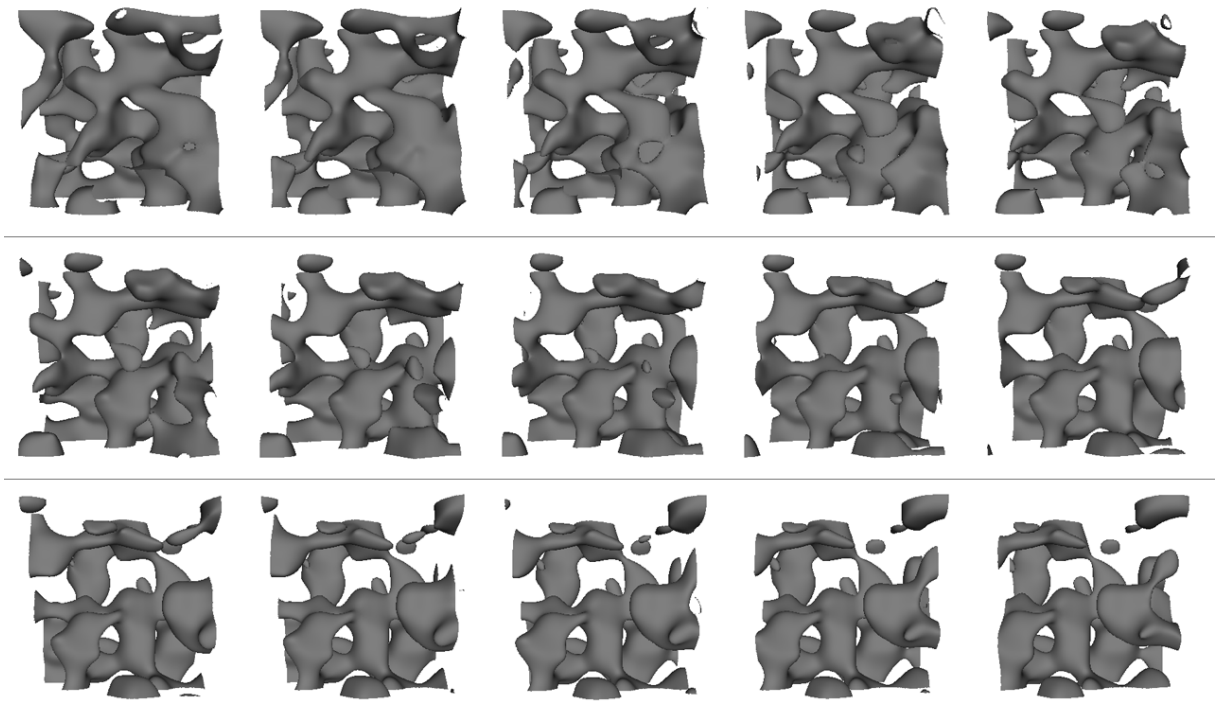


Figure 6: Generated meshes for Perlin noise dataset for 15 frames

Marc Alexa. Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[OR97] M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 59(4):365–385, 1997.

[TPG99] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved isosurface extraction. *Computers and Graphics*, 23:583–598, 1999.

[TSD07] Natalya Tatarchuk, Jeremy Shopf, and Christopher DeCoro. Real-time isosurface extraction using the gpu programmable geometry pipeline. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 122–137, New York, NY, USA, 2007. ACM.

[WKE99] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15:100–111, 1999.

A Layered Depth-of-Field Method for Solving Partial Occlusion

David C. Schedl*
david.schedl@cg.tuwien.ac.at

Michael Wimmer*
wimmer@cg.tuwien.ac.at

*Vienna University of Technology, Austria

ABSTRACT

Depth of field (DoF) represents a distance range around a focal plane, where objects on an image are crisp. DoF is one of the effects which significantly contributes to the photorealism of images and therefore is often simulated in rendered images. Various methods for simulating DoF have been proposed so far, but little tackle the issue of partial occlusion: Blurry objects near the camera are semi-transparent and result in partially visible background objects. This effect is strongly apparent in miniature and macro photography. In this work a DoF method is presented which simulates partial occlusion. The contribution of this work is a layered method where the scene is rendered into layers. Blurring is done efficiently with recursive Gaussian filters. Due to the usage of Gaussian filters big artifact-free blurring radii can be simulated at reasonable costs.

Keywords:
depth of field, rendering, real-time, layers, post-processing

1 INTRODUCTION

DoF represents a distance range around a focal plane in optic systems, such as camera lenses. Objects out of this range appear to be blurred compared to sharp objects in focus. This effect emphasizes objects in focus and therefore is an important artistic tool in pictures and videos.

People in the field of computer graphics aim for the ambitious goal of generating photo-realistic renderings. Depth of Field is one effect which significantly contributes to the photorealism of images because it is an effect that occurs in most optical systems. In computer renderings, the pinhole-camera model, which relies upon the assumption that all light-rays travel through one point before hitting the image plane, is used. Therefore, there is no focus range and no smearing occurs, resulting in a crisp image. However, in real-life optical systems—such as the eye or photographic cameras—sharp images are only produced if the viewed object is within a certain depth range: the depth of field.

DoF can be simulated very accurately by ray tracing, but the rendering of accurate DoF effects is far from interactive frame rates. For interactive applications, the effect has to be approximated in real time. Therefore, most approaches use fast post-processing techniques and sacrifice visual quality, causing artifacts. Common techniques to produce the DoF effect use an approach where pixels get smeared according to their circle of confusion (CoC) [25]. The CoC depends on the distance of objects and the lens parameters. One artifact in post-processing approaches is *partial occlusion*: An object in-focus occluded by an out-of-focus object should be partly visible at the blurred object borders of the front object. In computer graphics, the used pinhole camera model in combination with depth testing leads to a dismissing of background pixels. Real optical systems use a finite aperture camera model where light rays from occluded objects can hit the image sensor. Figure 1 shows this effect next to a pinhole rendering.

(a) (b)

Figure 1: A pinhole rendering of the scene *Dragons* resulting in a crisp image (a). Simulating shallow depth-of-field with the proposed method partly reveals occluded scene content (b). Note how the tongue of the dragon almost vanishes.

In this paper, we present an approach to tackling the partial occlusion problem. By rendering the scene with depth peeling [11], occluded pixels can be retrieved (Section 3.1). This occluded scene information is used

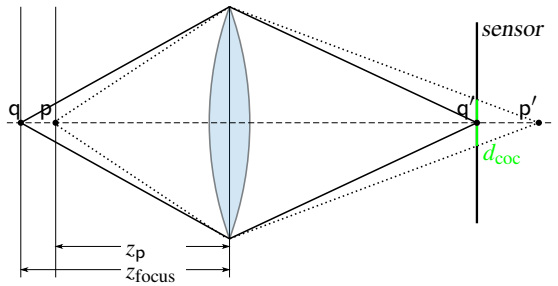


Figure 2: The model of a thin lens and how the points q (in-focus) and p (out-of-focus) are projected onto the image sensor (inspired by [25]).

to overcome the problem of partial occlusion. Rendered fragments are weighted, based on their depth, into layers (section 3.2), where each layer is blurred uniformly (section 3.3). Previous such layered DoF methods produced artifacts due to the layer splitting. We avoid most of these artifacts by smoothly decomposing layers and additional scene information. After blurring, the layers are composed by blending, thus producing renderings with convincing partial occlusion effects.

2 PREVIOUS WORK

DoF is an effect caused by the fact that optical lenses in camera systems refract light rays onto the image sensor, but fail to produce crisp projections for all rays. Figure 2 shows a schematics of a thin lens model and how rays are refracted. Although modern optical systems use a set of lenses, for the purpose of explaining DoF, a single lens is sufficient. Hypothetically, a sharp image point will only appear on the image plane from an object exactly in focus, located at z_{focus} (see figure 2). In practice, because of limitations of the human eye, objects within an acceptable sharpness are recognized as sharp. Objects out of the DoF range are projected as circles on the image plane. The diameter of this so-called circle of confusion (CoC) can be calculated as

$$d_{\text{coc}}(z,f,N,z_{\text{focus}}) = \left| \frac{f^2\,(z - z_{\text{focus}})}{zN\,(z_{\text{focus}} - f)} \right|, \qquad (1)$$

where z is the distance to the object in front of the lens, f is the focal length of the lens, N is the f-stop number, and z_{focus} is the focus distance [25].

While the blurring is produced as an imperfection in optical systems, computer renderings usually produce a crisp image. Therefore DoF has to be simulated by specific rendering methods [1, 2, 10, 3].

Methods operating in object space simulate rays that do not go through the center of the lens. These methods include distributed ray tracing [8] and the accumulation buffer method [12], which both produce high-quality results but fail to deliver real-time frame rates.

Faster methods are based on the idea of rendering the scene with a pinhole camera model and simulating the DoF effect via post processing, leading to few or no changes in the rendering pipeline. The first method discussed by Potmesil and Chakravarty in 1981 presented equation 1, the formula for calculating the CoC [25]. Most modern methods (including this one) are based on this work.

Methods using graphic cards for acceleration use pyramid methods, Poisson sampling or a combination of both [24, 26, 27, 13, 22]. Isotropic filters lead to intensity leaking artifacts, where colors from in-focus foreground pixel bleed on the out-of-focus background. Cross-bilateral filters or heat diffusion ([6, 14]) can be used to overcome this artifact, but this introduces other issues like discontinuity artifacts: Out-of-focus objects have sharp boundaries although the object itself is blurred.

The partial occlusion artifact is apparent in all previously mentioned methods. Rasterization techniques do not store occluded fragments, therefore it is not possible to accurately simulate transparency caused by out-of-focus smearing. To fully simulate this effect, occluded information has to be either stored or interpolated in layers. Layered DoF methods compose scene fragments into layers depending on fragment depth. With this representation it is possible to store or interpolate occluded scene content. Furthermore it is possible to uniformly blur each layer. One prominent artifact in layered DoF methods are discretization artifacts: Layers get blurred and therefore object borders are smeared out. When this smeared-out layer is blended with the other layers, the smeared border region appears as a ringing artifact at object borders due to the reduced opacity. In [4, 5], the authors investigate such artifacts. One way to avoid these ringing artifacts is presented in [18], where occluded scene information is interpolated before layers are created. Blurring and interpolation is done by a pyramidal method, which approximates a Gaussian filter. The author presents a variation of this method in [17], where the costly interpolation steps are left out and different filters are used. However, these methods fail to correctly solve the partial occlusion problem, because hidden scene information is only interpolated and does not represent any actual scene content.

The DoF methods [21, 15] are able to solve partial occlusion by generating CoC-sized splats for each pixel. However, these methods come with additional costs for sorting fragments, making them impractical for complex scenes.

In [19], layered rendering is used to generate a layered representation. The DoF effect is then generated by ray-traversing these layers. Rays are scattered across the aperture of a virtual lens, thus avoiding the previously mentioned discretization artifacts. An improvement is discussed in [20], where the layers are generated by

depth peeling and ray-tracing is done differently. Furthermore various lens effects (e.g., chromatic aberration and lens distortion) can be simulated. However, the method needs preprocessing previously to ray intersecting and needs back faces to be rendered. If the number of rays is not sufficient both methods produces noise and aliasing artifacts. Especially for strong blurs many rays have to be used, resulting in non-interactive rates.

For a solid approximation of partial occlusion, a layered scene representation, storing occluded fragments, has to be used. The approach presented in the following section is a layered DoF method which produces convincing partial occlusion effects while vastly avoiding the discussed issues.

3 METHOD

The method proposed in this paper decomposes the scene into *depth layers*, where each layer contains pixels of a certain depth range. The resulting layers are then blurred with a filter that is sized according to the distance from the focal point, and then composited. This approach handles partial occlusion, because hidden objects are represented in more distant layers and contribute to the compositing.

One way to generate the K layers would be to render the scene K times, with near- and far planes adjusted to cover the desired depth range of the layer. However, this leads to two problems: first, rendering the scene K times is too expensive for interactive applications, and second, discretization artifacts would appear due to the hard layer borders. In this paper, we solve both problems:

In order to avoid rendering the scene K times, we use depth peeling to generate a number M of *occlusion layers* (also named *buffers* in the following), where $M < K$. Note that each occlusion layer can contain fragments from the full depth range of the scene, while a depth layer is bound by its associated depth range. We then generate the depth layers by decomposing the occlusion layers into the depth ranges, which is much faster than rendering each depth layer separately.

To avoid discretization artifacts, we do not use hard boundaries for each depth layer, but a smooth transition between the layers, given by *matting functions*.

Furthermore, we also propose a method for efficiently computing both the blur and the layer composition in one step.

Our method consists of the following steps:

1. Render the scene into M buffers, where I_0 and Z_0 contain the color and depth from an initial pinhole rendering. The buffers $I_1 \dots I_{M-1}$ and $Z_1 \dots Z_{M-1}$ store peeled fragments from front to back.

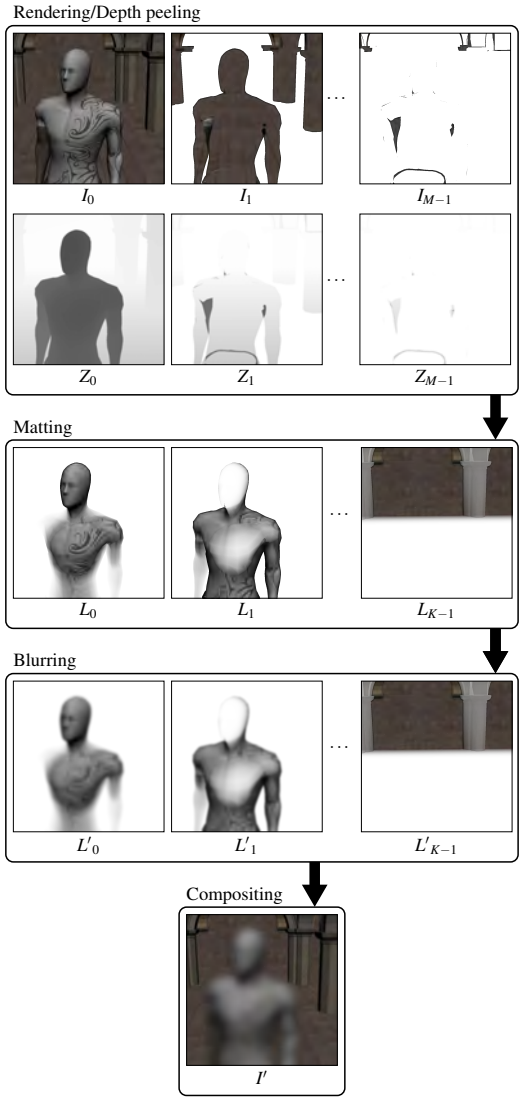


Figure 3: A overview of the proposed method in this work: The scene is rendered into color buffers I_0 to I_{M-1} and depth buffers Z_0 to Z_{M-1} by depth peeling . The color buffers are decomposed into K layers L_0 to L_{K-1} by a depth-dependent matting function. The decomposed layers get blurred by their CoC and composed. Note that the final algorithm combines the blurring and composing step, which is simplified in this figure.

2. Decompose the fragments of the input buffers into K depth layers L_0 to L_{K-1}, based on a matting function and the fragments' depth.

3. Blend and blur each layer L_k onto the buffers I'_{front}, I'_{focus} or I'_{back}, which are composed back-to-front afterwards.

Figure 3 outlines the above described algorithm.

3.1 Rendering

Rendering is done by depth peeling [11]. For depth peeling, first a 3D scene is rendered into a buffer storing the color I_0 and the depth Z_0 of a rendering, shown in figure 3. Then the scene is rendered a second time

into new buffers I_m and Z_m while projecting the previous depth buffer Z_{m-1} onto the scene. A fragment p gets rejected if its depth z_p has the same or smaller depth than the previously rendered fragment, stored in I_{m-1} and Z_{m-1}. This means that only previously occluded fragments are stored in I_m and Z_m. This is done iteratively until M layers are retrieved. If a fragment is rejected, it is "peeled away," revealing objects behind the first layer. Although there are faster peeling methods (e.g., [23]), we rely on [11], because peeling can be done iteratively from front to back.

3.2 Scene decomposition

The input images $I_0 \dots I_{M-1}$ are decomposed into K layers $L_0 \dots L_{(K-1)}$ by matting functions $\omega(z)$ and $\dot{\omega}$:

$$L_k = \Big(I_0 \cdot \omega_k(Z_0)\Big) \oplus \Big(I_1 \cdot \dot{\omega}_k(Z_1)\Big) \dots \oplus \Big(I_{M-1} \cdot \dot{\omega}_k(Z_{M-1})\Big). \quad (2)$$

The functions $\omega_k(z)$ and $\dot{\omega}_k(z)$ denote the matting function for the layer L_k and $A \oplus B$ denotes alpha-blending A over B.

3.2.1 Matting functions

The matting function ω_k was introduced in [18] and guarantees a smooth transition of objects between layers, while $\dot{\omega}_k$ retains a hard cut at the back layer boundaries to avoid situations where background fragments would be blended over foreground layers. The formulas are

$$\dot{\omega}_k(z) = \begin{cases} \frac{z - z_{k-2}}{z_{k-1} - z_{k-2}} & \text{for} \quad z_{k-2} < z < z_{k-1}, \\ 1 & \text{for} \quad z_{k-1} \le z \le z_k, \\ 0 & \text{otherwise}, \end{cases} \quad (3)$$

and

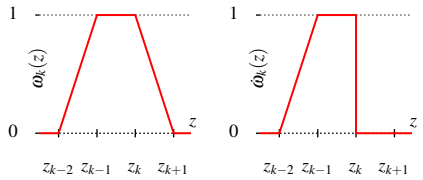
$$\omega_k(z) = \begin{cases} \frac{z_k - z}{z_k - z_{k+1}} & \text{for} \quad z_k < z < z_{k+1}, \\ \dot{\omega}_k(z) & \text{otherwise}, \end{cases} \quad (4)$$

where z_{k-2} to z_{k+1} defines anchor points for the layer boundaries. A plot of the functions is shown in figure 4. Special care has to be taken when matting the front L_0 and back L_{K-1} layer, where the boundaries are set to $z_{-2} = z_{-1} = z_0 = -\infty$ and $z_{K-1} = z_K = \infty$, respectively.

3.2.2 Layer boundaries

The layer matting relies on anchor points. Similarly to [18], the boundaries are spaced according to the filter size of the blurring method (further explained in section 3.3). Potmesil's formula for calculating the CoC (equation 1) can be rearranged to calculate a depth z based on a given CoC d. Since d_{coc} is non-injective, there are two possible results of this inversion:

$$d_{\text{coc}}^{-1}(d) = \big(D_1(d), D_2(d)\big) \quad (5)$$



(a) (b)

Figure 4: The matting functions ω_k (a) and $\dot{\omega}_k$ (b) with exemplary depth coordinates z_{k-2} to z_{k+1}.

with

$$D_1(d) = \frac{z_{\text{focus}} \cdot f^2}{f^2 + d \cdot N \cdot (z_{\text{focus}} - f)}, \quad (6)$$

$$D_2(d) = \frac{z_{\text{focus}} \cdot f^2}{f^2 - d \cdot N \cdot (z_{\text{focus}} - f)}. \quad (7)$$

With equation 5, the depth of anchor points can be calculated by using d_{coc}as input parameter, calculated by the filter size of the blurring method. Note that $D_2(d)$, $d \in \mathbb{R}^+$ is only applicable as long as

$$d < \frac{f^2}{N \cdot (z_{\text{focus}} - f)}. \quad (8)$$

The anchor point furthest away from the camera, z_{K-1}, is limited by this constraint. An anchor pointz_k is placed at the average CoC of the layers L_k and L_{k+1}. Thus

$$z_k = \begin{cases} D_1\left(\frac{d_k + d_{k+1}}{2}\right) & \text{for} \quad k < k_{\text{focus}}, \\ D_2\left(\frac{d_k + d_{k+1}}{2}\right) & \text{for} \quad k \ge k_{\text{focus}}, \end{cases} \quad (9)$$

where k_{focus} is the index of the layer in focus and d_k and d_{k+1} are the CoCs of the layers L_k and L_{k+1} respectively. The layer's CoC d_k is given by the blur radius for a discrete layer L_k, determined by the blurring method (see section 3.3).

3.2.3 Determining the number of layers

The depth of rendered fragments in the scene should lie within the depth range of the closest and furthest anchor points (z_{K-1} and z_0). Therefore enough anchor points to cover the scene have to be generated. This can be done manually or automatically. One naive automatic approach would be to use the near and far clipping planes, resulting in the highest possible depth range, which usually is not present in a scene. A better approach is to use hierarchical N-Buffers for determining the minimum and maximum depth values within the view frustum [9].

3.3 Blurring and Composition

We use Gaussian filters for blurring, because they can be separated, recursively applied and produce smooth results. The mapping from CoC to the standard deviation σ of a Gaussian kernel is chosen empirically as

$$d_{\text{pix}} = 4\sigma. \quad (10)$$

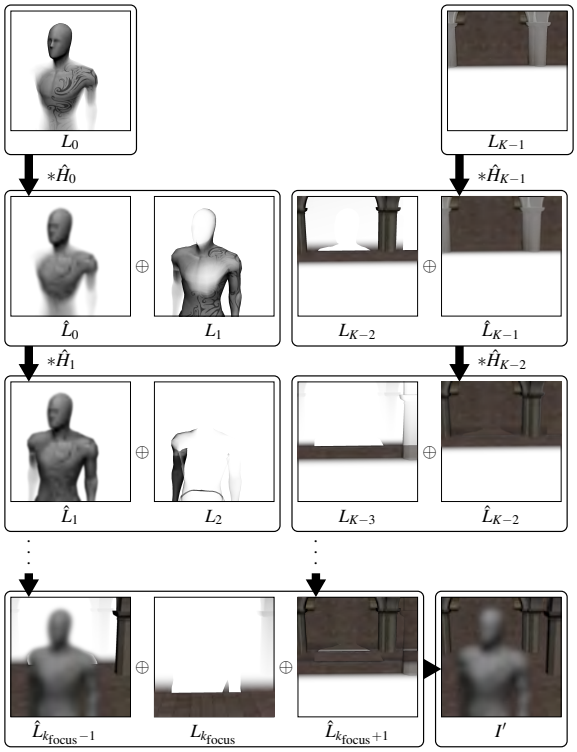


Figure 5: Overview of the combined blurring and composition steps: The layers L_0 to L_{K-1} are blended iteratively. Between each blending step the composition is blurred. Layers in front of the focus layer $L_{k_{\text{focus}}}$ and layers behind the focus layer are composed separately. Those are combined in a final step into the result I'.

Note that d_{pix} is the d_{coc} in screen coordinates and has to be transformed into the world coordinate system. Each layer is blurred by a Gaussian kernel H_k with the standard deviation σ_k as

$$L'_k = L_k * H_k, \tag{11}$$

where $*$ denotes a convolution.

Instead of convolving each layer separately (shown in figure 3 for illustrative reasons), a cascaded approach is chosen. Between each blur, one layer is blended onto the composition.

Layers in front and behind the in-focus layer have to be composed and blurred separately. Otherwise it is not possible to keep the correct depth ordering of the layers. The composition of the front layer starts by taking the layer closest to the camera (i.e., L_0) and blurring it with the filter kernel $\hat{H}_0$. In the next step this blurred layer is blended over the next closest layer (i.e., L_1) and afterwards blurred with $\hat{H}_1$. A schematic of the composition steps is shown in figure 5. Since a blurred layer L_k is blended over L_{k+1} and then blurred again, the effect of this method is that L_k is blurred by $\hat{H}_k$ and by $\hat{H}_{k+1}$. The iteration continues until the layer in-focus $L_{k_{\text{focus}}}$ is reached. In general, such recursive Gaussian filters produce the same result as blurring with one big Gaussian. The resulting filter sizes can be calculated by the Euclidean distance [7, chapter 8]. However, in our application the results differ due to occlusions within the layers.

Back layers are blurred similarly, starting with L_{K-1}. To keep the correct layer ordering, the layer closer to the camera (i.e., L_{K-2}) has to be blended over the previously blurred layer. The iteration is again continued until the layer in-focus is reached.

The number of blurring iterations for a layer L_k is given by $|k - k_{\text{focus}}|$. Calculating the final composition I' is done by

$$I' = \hat{L}_{k_{\text{focus}}-1} \oplus (L_{k_{\text{focus}}} \oplus \hat{L}_{k_{\text{focus}}+1}), \tag{12}$$

where

$$\hat{L}_k = \begin{cases} L_k & \text{for} \quad k = k_{\text{focus}} \\ L_k * \hat{H}_k & \text{for} \quad k = 0 \text{ and } k = K-1 \\ (\hat{L}_{k-1} \oplus L_k) * \hat{H}_k & \text{for} \quad k < k_{\text{focus}} \\ (L_k \oplus \hat{L}_{k+1}) * \hat{H}_k & \text{for} \quad k > k_{\text{focus}} \end{cases} \tag{13}$$

Results in section 4 are produced with a Gaussian filter kernel $\hat{H}_k$ with a standard deviation of $\hat{\sigma}_k$:

$$\hat{\sigma}_k = |k - k_{\text{focus}}|. \tag{14}$$

Various methods for calculating the filter size can be used. For Gaussians, the adequate (non-recursive) σ_k can be calculated by

$$\sigma_k = \begin{cases} 0 & \text{for} \quad k = k_{\text{focus}}, \\ \sqrt{\hat{\sigma}_k^2 + \sigma_{k+1}^2} & \text{for} \quad k < k_{\text{focus}}, \\ \sqrt{\hat{\sigma}_k^2 + \sigma_{k-1}^2} & \text{for} \quad k > k_{\text{focus}}, \end{cases} \tag{15}$$

where k is in the interval $[0, K-1]$.

3.3.1 *Normalization*

Due to the usage of matting functions ω and $\dot{\omega}$, resulting in expanded depth layers, and the usage of depth peeling, discretization artifacts as discussed in [5, 4] are mostly avoided. However, in some circumstances (e.g., almost perpendicular planes) such artifacts may still appear. We use premultiplied color values while matting and filtering. Therefore the composition can be normalized (divided by alpha), thus further minimizing discretization artifacts.

4 RESULTS

The proposed method was implemented in OpenGL and the shading language GLSL. Performance benchmarks are done on an Intel Core i7 920 CPU with a Geforce GTX 480 graphics card. Depth peeling uses a 32bit

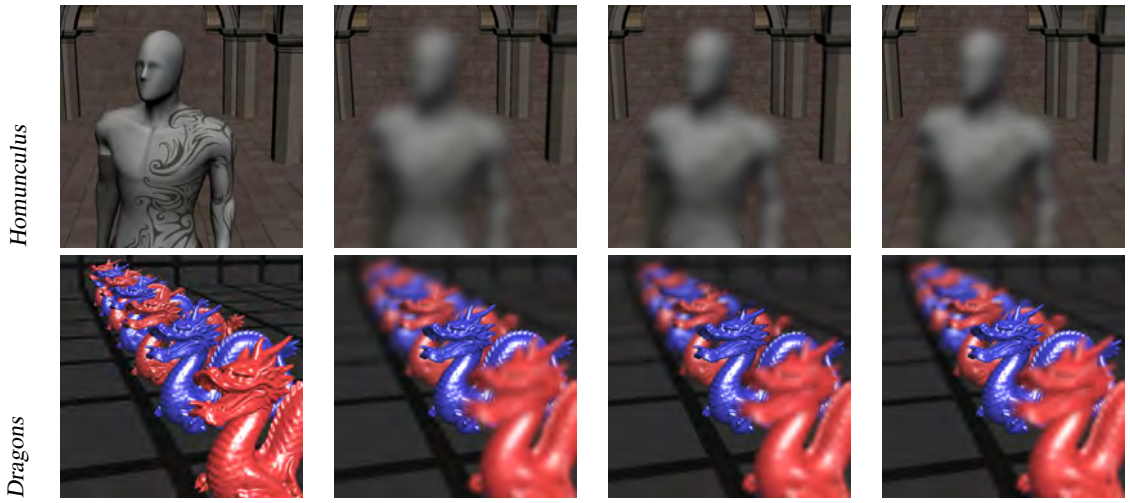


Figure 6: DoF effects produced with our method, the ray-traversal method ([20]) and the accumulation-buffer method. The first row shows the scene *Homunculus* (74k faces), and the second shows *Dragons* (610k faces). Renderings have the resolution 1024×1024. Note that there are sampling artifacts on the chest of the *Homunculus* scene in the accumulation and ray-traversal method, although there are 256 rays/views used. Our method avoids such artifacts by cascaded Gaussian filtering. Also note the partial-occlusion artifacts (e.g., in the second red dragon) in the ray-traversal method. The lens settings are $f = 0.1$, $N = 1.4$ and is focused at the stone wall in the back ($z_{\text{focus}} = 18.5$) for the *Homunculus* and at the foremost blue dragon ($z_{\text{focus}} = 3$) for the *Dragon* scene.

z-buffer to avoid any bias values due to z-buffer imprecisions.

We compare our method to the accumulation buffer-technique [12] and to a ray-traversal technique[20], because the first simulates high-quality DoF effects if enough views are sampled, while the latter method is a state-of-the-art method which handles partial occlusion correctly. The accumulation buffer technique is implemented in OpenGL, but does not use the accumulation buffer, because of precision issues when accumulating a high number of views. Instead, a 32bit-per-channel float texture is used for accumulation. The ray-traversal method was implemented in OpenGL and GLSL. Although there are some tweakable parameters, the authors give little information about their configuration. So for the intersection tests of rays, we use 100 steps for the linear search and 5 for the binary search in normalized device coordinates. A hierarchical N-Buffer is used to decrease the depth range for each rays. In our implementation, we decrease the depth range for each ray individually—while the original authors packed rays—and we use 4 regular depth-peeling layers, without any optimizations. Additionally, the ray-traversal method requires closed scene objects and back-face culling to be turned off, for reliable intersection testing. This introduces additional rendering costs and decreases the usable peeling layers to only 2. The authors propose ways to overcome the latter limitation. However, in our implementation we use the simplified version containing only two peeling layers and their backface counterparts. For both reference methods we use a Gaussian distribution for lens samples positioning.

The methods are applied to the scenes *Homunculus* (74k triangles) and *Dragons* (610k triangles), shown in figure 6. Performance comparisons are shown in table 1. Rendering costs for the accumulation-buffer method are basically the costs for one scene rendering multiplied by the number of views. Our method is, apart from depth peeling, independent of the scene complexity, and faster than the ray-traversal method, even when that method uses only 32 rays, resulting in sampling artifacts. Our method is faster at processing the *Dragons* scene, although the scene *Homunculus* has fewer triangles. This can be explained by the distribution of the depth layers and the resulting amount of blur. In the *Homunculus* scene there are more highly blurred foreground layers, resulting in overall more rendering costs than in *Dragons*, where the layers are more, but evenly spread. Note that although scene *Dragons* has more triangles than *Homunculus*, it is rendered faster due to shading without textures and the use of vertex-buffer objects.

We currently store all sub-images on the graphics card—for convenience and debug reasons—resulting in heavy memory usage. However, additional depth layers ($\hat{L}_0$ to $\hat{L}_{K-1}$) can be avoided by applying the process-queue (matting, blurring, composing) in one buffer, which would decrease memory consumption.

	our (DP/matting/blur) Total		(DP/ray-traversal) Total			accum.
	cascaded	non-cascaded	256	128	32 rays	256 views
Homunculus (74k tri.)	(46/5/51)102	(46/5/95)146	(58/1290)1348	(48/643)691	(48/140)188	4809
Dragons (610k tri.)	(40/7/51)98	(40/8/85)133	(69/1374)1443	(69/685)754	(59/152)211	4163

Table 1: Performance comparisons, in ms, of our method (cascaded and non-cascaded blurring) with the ray-traversal method ([20]) and the accumulation-buffer method for the scenes *Homunculus* and *Dragons*. Renderings have the resolution 1024×1024 and 4 Depth-peeling iterations (DP) have been used.

5 CONCLUSION AND FUTURE WORK

We have presented a depth-of-field post-processing method with the aim of overcoming the partial occlusion artifact. The contribution of this work is a simple, efficient and GPU-friendly method. We combine depth-peeling with improved matting functions to avoid the overhead of rendering to a high number of depth layers. Furthermore we use high-quality Gaussian filters in a recursive way, which has not been done—to our knowledge—in DoF methods before. With the usage of Gaussian filters, high blur radii can be simulated, where even the reference methods start to produce sampling artifacts. We have shown that those DoF effects can be produced at frame rates that are significantly higher than previous methods, making high-quality DoF available for interactive applications.

One important step for the correct handling of partial occlusion is depth peeling, which is frequently used to resolve transparency issues, thus making the method hardly usable for interactive applications like games.

Currently we use Gaussian filters, which are separable and can be computed efficiently while delivering artifacts-free images. The usage of cascaded filters while composing the DoF effect slightly alters the produced image, but results in better performance. If higher frame rates are needed and visual quality can be sacrificed faster blurring methods (e.g., box, pyramid filters) can be used.

The composition by alpha-blending is simple and efficient, thus leading to faster results when compared to current methods like [20]. Layering discretization artifacts known from other methods are mostly avoided by matting, depth peeling and normalization.

Wide-spread anti-aliasing methods (i.e., MSAA) cannot be easily enabled for our method. However, image-based anti-aliasing methods (such as MLAA or FXAA)—which are becoming more popular due to the wide usage of deferred shading—can be applied.

Currently, layers are split based on the d_{coc} of fragments and on the chosen blurring method. This might result in empty layers. Decomposition could be optimized by using clustering methods, such as k-means clustering, as proposed in [21, 16]. With the use of clustering, layer borders could be tailored to the pixel density in scenes and empty layers could be avoided. However, clustering is a costly process and therefore only applicable for off-line rendering.

One further field of investigation would be the impact of correct partial occlusion rendering on human perception. We think that a correct handling of partial occlusion in combination with gaze-dependent focusing (e.g., with an eye-tracker) would result in deeper immersion of the user.

6 ACKNOWLEDGMENTS

Thanks to Juergen Koller for providing the Homunculus model. The used Dragon and Sponza models are courtesy of Stanford Computer Graphics Laboratory and Marko Dabrovic.

7 REFERENCES

[1] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu. Camera models and optical systems used in computer graphics: Part I, Object based techniques. Technical report, University of Berkeley, California, USA, 2003.

[2] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu. Camera models and optical systems used in computer graphics: Part II, Image-based techniques. Technical report, University of Berkeley, California, USA, 2003.

[3] Brian A. Barsky and Todd J. Kosloff. Algorithms for rendering depth of field effects in computer graphics. In *Proceedings of the 12th WSEAS international conference on Computers*, pages 999–1010, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

[4] Brian A. Barsky, Daniel R. Tobias, Michael J.and Horn, and Derrick P. Chu. Investigating occlusion and discretization problems in image space blurring techniques. In *First International Conference on Vision, Video and Graphics*, pages 97–102, University of Bath, UK, July 2003.

[5] Brian A. Barsky, Michael J. Tobias, Derrick P. Chu, and Daniel R. Horn. Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. *Graphics Models*, 67(6):584–599, November 2005.

[6] Marcelo Bertalmio, Pere Fort, and Daniel Sanchez-Crespo. Real-time accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, 3DPVT '04, pages 767–773, Washington, DC, USA, 2004.

[7] Wilhelm Burger and Mark J. Burge. Principles of Digital Image Processing: Advanced Techniques. To appear, 2011.

[8] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 137–145, New York, NY, USA, 1984. ACM.

[9] Xavier Décoret. N-buffers for efficient depth map query. *Computer Graphics Forum*, 24(3), 2005.

[10] J. Demers. Depth of field: A survey of techniques. In Fernand Randima, editor, *GPU Gems*, chapter 23, pages 375–390. Pearson Education, 2004.

[11] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA, 2001.

[12] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. *SIGGRAPH Computer Graphics*, 24:309–318, September 1990.

[13] Earl J. Hammon. Practical post-process depth of field. In Hubert Nguyen, editor, *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 28, pages 583–606. Addison-Wesley, 2007.

[14] Michael Kass, Lefohn Aaron, and John Owens. Interactive depth of field using simulated diffusion on a GPU. Technical report, Pixar Animation Studios, 2006.

[15] Todd J. Kosloff, Michael W. Tao, and Brian A. Barsky. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proceedings of Graphics Interface 2009*, pages 39–46, Toronto, Canada, 2009.

[16] Todd Jerome Kosloff. *Fast Image Filters for Depth of Field Post-Processing*. PhD thesis, EECS Department, University of California, Berkeley, May 2010.

[17] Martin Kraus. Using Opaque Image Blur for Real-Time Depth-of-Field Rendering. In *Proceedings of the International Conference on Computer Graphics Theory and Applications : GRAPP 2011*, pages 153–159, Portugal, 2011. Institute for Systems and Technologies of Information, Control and Communication.

[18] Martin Kraus and Magnus Strengert. Depth-of-field rendering by pyramidal image processing. *Computer Graphics Forum*, 26(3):645–654, 2007.

[19] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (TOG)*, 28(5):1–6, 2009.

[20] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. *ACM Transactions on Graphics (TOG)*, 29(4):65:1–65:7, July 2010.

[21] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using splatting on per-pixel layers. *Computer Graphics Forum (Proc. Pacific Graphics'08)*, 27(7):1955–1962, 2008.

[22] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):453–464, 2009.

[23] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Single pass depth peeling via cuda rasterizer. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, New York, NY, USA, 2009. ACM.

[24] Jurriaan D. Mulder and Robert van Liere. Fast perception-based depth of field rendering. In *Proceedings of the ACM symposium on Virtual Reality Software and Technology*, VRST '00, pages 129–133, Seoul, Korea, October 2000. ACM.

[25] Michael Potmesil and Indranil Chakravarty. A lens and aperture camera model for synthetic image generation. In *Proceedings of the 8th Aannual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '81, pages 297–305, Dallas, Texas, USA, 1981. ACM.

[26] Guennadi Riguer. Real-time depth of field simulation. In Wolfgang F. Engel, editor, *ShaderX^2: Shader Programming Tips and Tricks with DirectX 9*, pages 529–556. Wordware Publishing, October 2003.

[27] Thorsten Scheuermann and Natalya Tatarchuk. Improved depth of field rendering. In Wolfgang Engel, editor, *ShaderX^3: Advanced Rendering Techniques in DirectX and OpenGL*, pages 363–378. Charles River Media, 2004.

Journal of WSCG

Index