Proceedings of the

16th Central European Seminar on Computer Graphics

April 29 - May 1, 2012
Smolenice, Slovakia
Co-organized with SCCG

Institute of Computer Graphics and Algorithms
Vienna University of Technology

Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava

Sponsors

vrvis

OESTERREICHISCHE COMPUTER GESELLSCHAFT
AUSTRIAN COMPUTER SOCIETY

PC REVUE

The Ministry of Education of the Slovak Republic

Slovak Society of Computer Science

Eurographics

sféra
Grafické informačné systémy

Edited by Michael Wimmer, Jiří Hladůvka, and Martin Ilčík © 2012
ISBN:978-3-9502533-4-4

Impressum

Vienna University of Technology
Institute of Computer Graphics and Algorithms
Favoritenstraße 9-11/186
1040 Vienna

ISBN 978-3-9502533-4-4

Welcome to CESCG 2012!

This book contains the proceedings of the 16th Central European Seminar on Computer Graphics, short CESCG, which continues a history of very successful seminars. Again this year, CESCG proceedings have an ISBN (978-3-9502533-4-4) and will therefore remain retrievable as long as there are libraries!

The long history of CESCG has started in 1997 in a medium-sized lecture room in Bratislava, bringing together students from Bratislava, Brno, Budapest, Graz, Prague, and Vienna. The idea found wide appraisal and the seminar moved to the beautiful castle of Budmerice, where it was held for 8 consecutive years, constantly growing in size and attraction. It was just in the 10th anniversary year 2006 that CESCG had to take a detour to move to Častá-Papiernička Centre, while it was back in Budmerice castle since 2007. Unfortunately, since 2011 the Budmerice castle is not available for scientific activities. After spending the previous year in Viničné, for this year we are moving to the beautiful castle in Smolenice.

Who are the CESCG heroes who made this year's seminar happen? In no particular order – because many people were involved equally – we would like to thank the organizers from Vienna: Michael Wimmer, Anita Mayerhofer, Werner Purgathofer, and especially Martin Ilčík for taking care of the complete reviewing process and scientific program preparation. We are very thankful to the CESCG organizers from Bratislava, mainly Andrej Ferko, always an inspiration to CESCG; and Matej Novotný, Ela Šikudová, David Běhal, Janka Dadová, Roman Ďurikovič, and Ján Lacko for the excellent preparations and on-site organization.

The main idea of CESCG is to bring students of computer graphics together across boundaries of universities and countries. Therefore we are proud to state that we have achieved again a very high number of 15 participating institutions and a very tight time schedule of 27 valuable student works and two invited talks. We welcome groups from Bratislava, Slovakia; Brno (VUT and MU), Plzeň and Prague (CTU and KU), Czech Republic; Budapest, Hungary; Bonn, Germany; Graz and Vienna, Austria; Szczecin, Poland; Bergen, Norway; Maribor, Slovenia; and Sarajevo (Univ. and SSST), Bosnia and Herzegovina.

We assembled a large International Program Committee of 21 members, allowing us to have each paper reviewed by three IPC members during the informal reviewing process. We would like to thank the members of the IPC for their contribution to the reviewing process. The IPC of CESCG 2012 consists of:

Jiří Bittner	Ivana Kolingerová	Marc Streit
Alan Chalmers	Radoslaw Mantiuk	László Szirmay-Kalos
Silvester Czanner	Stephan Mantler	Veronika Šoltészová
Andrej Ferko	Jozef Pelikán	Ania Tomaszewska
Jasminka Hasić	Selma Rizvić	Michael Wimmer
Helwig Hauser	Jiří Sochor	Pavel Zemčík
Reinhard Klein	Markus Steinberger	Borut Žalik

The first invited talk "Having Fun With Tables: Research into novel interfaces – two dimensions and above" will be held by Joaquim A. Jorge from Departamento de Engenharia Informática of Instituto Superior Técnico, Portugal. The second invited talk by Carol O'Sullivan from School of Computer Science and Statistics of Trinity College Dublin, Ireland, will be about "Perceiving Realism in Virtual Worlds".

The seminar is held under the auspices of the Austrian Ambassador to Slovakia, His Excellency Dr. Josef Marcus Wuketich, and is co-organized with the Spring Conference on Computer Graphics (SCCG), which takes place right after the seminar.

The organization of a seminar where there are only low expenses for the students requires funding. We are very thankful to the sponsors of CESCG 2012:

- VRVis, a research center for virtual reality and visualization in Vienna
- OCG, the Austrian Computer Association
- SISp, Slovak Society for Computer Science
- Sféra, graphical information systems
- Eurographics, the European Association for Computer Graphics
- The Ministry of Education, Science, Research and Sport of the Slovak Republic
- PC Revue, a slovak computer magazine

Please note that the electronic version of these proceedings is also available at http://www.cescg.org/CESCG-2012/.

Vienna, April 2012

Michael Wimmer
Jiří Hladůvka
Martin Ilčík

Table of Contents

Invited Talks

Computational Geometry

Applications & Image Recognition

3D Data Processing

Cameras & Materials

Natural Phenomena & Perception

Visualization

Poster Session

Color Plates

Sponsors of CESCG 2012

Invited Talks

Having Fun With Tables: Research into novel interfaces – two dimensions and above

Joaquim A. Jorge

Instituto Superior Técnico
Portugal

Abstract

Work on interactive tabletops and surfaces has focused mostly on two-dimensional issues, such as multi-finger gestures and tangible interaction. Interesting as it is, however this picture is missing several dimensions. I will describe work on 2D and 3D semi-immersive environments and present novel on-and-above-the-surface techniques based on bi-manual models to take advantage of the continuous interaction space for creating and editing 3D models in stereoscopic environments. I will also discuss means to allow for more expressive interactions, including novel uses of sound and combining hand and finger tracking in the space above the table with multitouch gestures on its surface continuously. These combinations can provide alternative design environments and allow novel interaction modalities.

Bibliographical Details Joaquim Jorge is a Professor at Instituto Superior Técnico (IST/UTL), the School of Engineering of the Technical University of Lisboa, Portugal, where he teaches User Interfaces and Computer Graphics. He received PhD and MSc degrees in Computer Science from Rensselaer Polytechnic Institute, Troy, NY, in 1995. He is Editor in Chief of Computers & Graphics Journal and a member of the ERCIM Editorial Board. He is a member of ACM/SIGGRAPH, IEEE Computer Society, IFIP TC13 (Human Computer Interaction). He has also served on the EG Education Board since its inception in 2001 until 2011. Joaquim Jorge's interests are in Calligraphic and Multimodal User Interfaces, Visual Languages and Pattern Recognition techniques applied to Human-Computer Interaction. He was elected Fellow of the Eurographics Association in 2010.

Perceiving Realism in Virtual Worlds

Carol O'Sullivan

Trinity College Dublin
Ireland

Abstract

We may not even realise it, but Virtual Worlds are becoming more and more a part of our lives. We see them in movies, video games, online communities and even in serious applications, such as health and education. Research in the field of Visual Computing has contributed greatly to increasing the realism of virtual objects, scenes and characters, by drawing on fundamental mathematical, scientific and technical principles to create stunning visual effects. Ultimately, however, the realism of a virtual world is in the eye of the beholder, so human perception must also be considered as an integral part of the creative process. In this talk, I will discuss some of the challenges of creating compelling dynamic scenes, from simple colliding spheres to a complex populated Metropolis.
Bibliographical Details

Bibliographical Details Carol O'Sullivan is the Professor of Visual Computing in the school of Computer Science in Trinity, where she leads the Graphics, Vision and Visualisation group (GV2) and is the Director of the newly established Centre for Creative Technologies. After receiving a B.A. in Mathematics from Trinity College in 1988, she worked for several years as a software engineer in industry (mainly in Germany), followed by a Masters degree from Dublin City University in 1996 and a PhD in computer graphics from TCD in 1999. She has published over 120 peer-reviewed papers and supervised over 20 PhD students since then. She was elected as the first Irish Fellow of the European Association for Computer Graphics (Eurographics) in 2007, and as a Fellow of Trinity College in 2003. She is the co-Editor in Chief of the ACM Transactions on Applied Perception, and the Associate Editor in Chief for Special Issues of IEEE Computer Graphics and Applications. She has chaired several international conferences, including Eurographics 2005, and has been a member of many international program committees, including the SIGGRAPH and Eurographics papers committees.

Computational Geometry

Progressive Hulls: Application on Biomedical Data

David Cholt*
Supervised by: Josef Kohout†

Department of Computer Science and Engineering
University of West Bohemia
Pilsen / Czech Republic

Abstract

A coarse outer hull of a mesh is a good tool used in computer graphics to reduce algorithm complexity, especially, in applications such as collision detection or ray-tracing. It is often required that the hull has some very specific parameters concerning its shape and quality since these influence general flexibility and numerical stability of the algorithms. This paper overviews existing problem approaches, their downsides for coarse outer hull creation, and describes *Progressive Hull* algorithm, which produces coarse hulls that maintain the general shape of the original triangulated mesh while containing the original mesh inside its interior. However, when this algorithm is used on a biomedical mesh extracted from volumetric data, one can observe frequent artifacts in the produced hull caused by local imperfections in the meshes. In this paper, we, therefore, present a few modifications to the original *Progressive Hull* algorithm that result not only in a suppression of hull artifacts and a better overall hull quality but also in a shorter execution time.

Keywords: Progressive Hull, Coarse Hull, Outer Hull, Mesh Decimation, Progressive Hull Application

1 Introduction

A coarse outer hull of a mesh is any hull that encapsulates the mesh completely and has a lower number of primitives (vertices, triangles) and, if possible, preserves the shape of the mesh. It can be used in many applications, for example in ray-tracing where one can detect the possibility of a ray intersecting the mesh by finding the intersection with the hull first. Given that the hull contains less polygons than the original mesh, the hull intersection test is faster. Furthermore it is more precise than tests using bounding box or convex hull, since the shape of the coarse outer hull is more similar to the shape of the mesh.

In our case we exploited the properties of the hull in in our project aiming at a simulation of human musculoskeletal system. Every muscle in this system is represented by a triangulated surface mesh which is wrapped around bones and gets deformed as these bones move. The deformation method, which is based on gradient domain deformation technique and described in [4], requires a very specific coarse outer hull of the muscle mesh as its input in order to speed up the deformation process. The deformation is performed on the hull and projected on the muscle mesh using barycentric coordinates that were previously constructed. This allows the method to be more numerically stable and faster. The coarse hull of the mesh therefore must meet following criteria:

- Low primitive count compared to original mesh - the hull has to be simple enough, so that later complex computations can be performed on a limited number of primitives. Less than $\frac{1}{10}$ of the original primitive count is desired
- Outer hull - all primitives of the hull must be outside of the mesh, leaving some spacing between the mesh and the hull. This is due to barycentric coordinates requirements
- Shape preservation - the hull has to ”trace” the shape of the original mesh. In other words, the spacing between the original mesh and the hull has to be consistent
- Non-self-intersecting hull - edges and triangles of the hull may not intersect the hull as this would cause instability in deformation computation
- The hull has to be manifold and should be smooth (depending on the muscle data), otherwise this would cause deformation instability as well

Outline of this paper follows. Section 2 describes examples of methods used to obtain coarse outer hulls and their disadvantages, Section 3 describes *Progressive Hull* algorithm, which should produce hulls that meet the criteria above, Section 4 introduces our modifications of the algorithm for better results on our biomedical data, followed by Sections 5 presenting experimental results, a hull quality comparison and execution time measurements. Our paper is concluded in Section 6.

*cholt@students.zcu.cz
†besoft@kiv.zcu.cz

2 State of the art

In this section we briefly describe the methods, that can be used to obtain a coarse outer hull of a triangulated mesh. Since we use the coarse hull in the deformation method described in [4], here we discuss some possible approaches to obtain a specific hull that meets all the criteria described in Section 1. This paper does not provide an overview of methods used in collision detection, raytracing, silhouette clipping or another methods that use the coarse outer hull as we did not use it for those purposes.

Bounding box, as an example of a very simple coarse hull, does not preserve the shape of the original mesh. Convex hull meets more of the criteria specified, but the spacing between the mesh and the hull varies significantly, especially if the mesh is rugged.

Better approach to the problem is to create an *alpha shape* [2], which is an object created from a finite set of vertices, in our case the mesh vertices. For the sake of simplicity, we will describe the construction and problems of an *alpha shape* in two dimensions. The method has one tuning parameter α that defines a radius of an abstract disc. The method finds such discs that have the property that two of the vertices lie on their boundary and they do not contain any other vertices. Vertices of the *alpha shape* are generated at the intersection of two neighboring discs. The situation is similar for three-dimensional space, only one has to use spheres instead of discs and search for intersections of three neighboring spheres. We can additionally implement a restriction that the intersection must lie outside of the original mesh in order to achieve outer hull. This method for finding an "*alpha hull*" from a set of points allows one to create an object, that is not necessarily convex and therefore, to certain extent, resembles the shape of the original mesh. However, this approach has four main problems:

1. If the α parameter is too large, the resulting hull suffers from the same problem as convex hull, i.e. the spacing between the alpha hull and the mesh varies significantly (for $\alpha \to \infty$ the alpha hull is equal to convex hull; see Fig. 1a)
2. If the α parameter is too small, the hull would intersect the original mesh (see Fig. 1b). Very small α parameter also causes the hull to be divided into number of components that can not form the hull of the original mesh by definition.
3. Some (arguably important) details in the original mesh may be lost in the process (Fig. 1a)
4. Even if we would be able find an optimal α parameter that ensures the *alpha hull* is a coarse outer hull of the object, the primitive count of the hull would be too high, approximately the same as the primitive count of the original mesh. In general, the number of primitives in resulting hull cannot be controlled well enough in *alpha shapes*

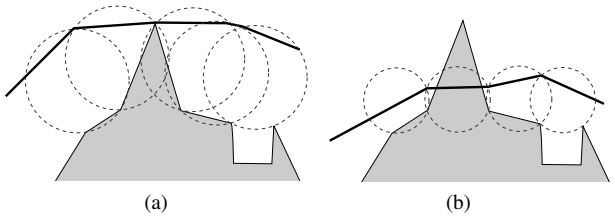


Figure 1: (a) *alpha shape* with too large α parameter. (b) *alpha shape* with too small α parameter.

Another possible approach is to decimate the mesh and therefore obtain a coarse mesh directly from the original mesh. Decimated mesh maintains the shape of the mesh by definition and has desired lower primitive count. One can assume that enlarging the mesh by moving its vertices outwards in the direction of their surface normal would create a coarse outer hull.

The problem is how much we need to enlarge the decimated mesh. If we enlarge the mesh too much, self-intersections may occur (see Fig. 2 for example). However, if we do not enlarge it enough, the decimated mesh would intersect the original mesh, as many decimation algorithms are based on volume preservation.

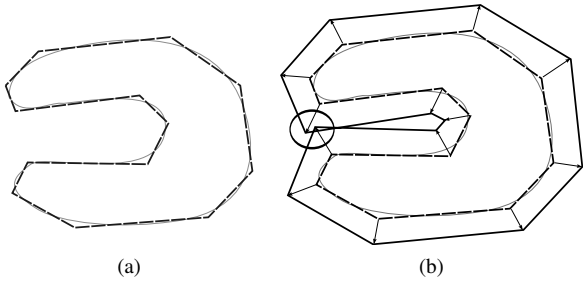


Figure 2: (a) Decimated mesh. (b) Enlarged decimated mesh with self-intersection introduced.

Nevertheless, this approach is simple to implement and has a sufficient shape and primitive count control, though it may give inconsistent results. We need an algorithm that creates an outer hull of the mesh by decimating and enlarging it at the same time, with a better control of how the decimation is performed.

3 Progressive hull

Progressive Hull [7] is a generalized mesh simplification process that meets all the criteria described in Section 1. The method is based on decimation of the mesh by a sequence of edge collapses, that ensures that progressively created hull contains the whole original mesh in its interior volume.

Figure 3 shows how the edge **e**, surrounded by faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$ and defined by vertices V_{e1} and V_{e2} collapses. Vertices V_{e1} and V_{e2} are joined

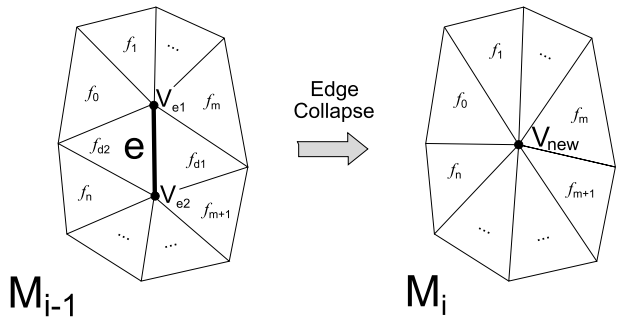


Figure 3: Edge collapse

by this operation to form a new vertex V_{new} and affected primitives are adjusted accordingly. Note that faces f_{d1} and f_{d2} are removed from the mesh and therefore every collapse reduces the number of faces in the mesh by two.

In order for the sequence of such collapses to result in a progressive hull, we need to calculate a specific position for the vertex V_{new}. Paper [7] shows that in order for the mesh M_i (after one edge collapse) in every iteration i to be an outer hull of the mesh M_{i-1} (before the collapse), the volume of the mesh M_i must be greater or equal to the volume of the mesh M_{i-1}. This can be achieved by placing the vertex V_{new} inside the intersection of half spaces above faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$ (see Fig. 4 for simplified two-dimensional case).

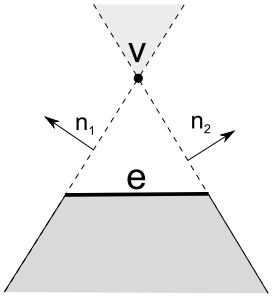


Figure 4: Position of vertex **V** constrained to lie in intersection of half spaces defined by planes with normals n_1 and n_2

The volumes of the meshes M_{i-1} and M_i are computed as a sum of the tetrahedral volumes defined by each mesh triangle's vertices and the origin point. Since we need the hull to be similar to the original mesh as much as possible, we place the vertex V_{new} in a way that it causes the smallest possible volume gain. That is a linear programming problem with an objective of mesh volume gain minimization and constrains defined by equations of half spaces above faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$.

Every mesh edge enters a priority queue with a priority based on the volume gain introduced by its collapse. The lower is the change, the higher is the priority. The algorithm follows:

1. For every edge in the mesh, compute the volume change that would be introduced by collapsing the edge. That requires solving linear programming problem. Store the solution for later use.
2. Insert the edge to a priority queue, low mesh volume gain represents a high priority in the queue
3. While the queue is not empty and the target primitive count is not reached:
   (a) Remove the edge from the priority queue and collapse it
   (b) Recalculate priority of every edge that was affected by this collapse and update their position in the priority queue by solving the linear programming problem again

One can see that this algorithm can be quite slow. Many of the linear programming problem solutions are redundant, as they are invalidated by a later nearby edge collapse. Additionally, a time-consuming solution of the linear programming problem is required for every edge priority update.

Platis and Theoharis [6] suggested using a faster approach to priority computation. Instead of using vertex V_{new} computed by solving the linear programming problem to determine the volume of the mesh M_i after the edge collapses, they proposed using an arithmetic average of the vertices in the one ring area surrounding the collapsing edge, V_{avg}.

The volume computed using this average scales similarly to the volume originally computed using vertex V_{new}, i.e. in relatively flat area adjacent to the edge **e**, the volume gain caused by imaginary collapse into the vertex V_{avg} is lower than in rugged area and therefore the induced priority is higher. This approach is considerably faster than solving the linear programming problem.

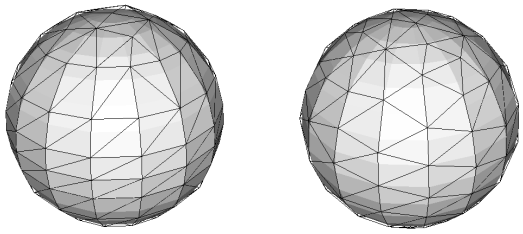
4 Modified progressive hull

Papers [7] and [6] do not mention any major problems regarding the quality of the resulting progressive hulls. However, our implementation of described methods performed irregular hull construction, showed significant numerical instability and produced low quality triangles in the hull. This was caused by imperfections in our meshes, presence of nearly or fully parallel triangles, local non-manifold areas in the mesh and other problems, which were probably not considered in [7] and [6]. Therefore, we introduced several modifications to the algorithm in order to resolve these issues.

4.1 Volume increase computation

The original paper [7] suggests the volume of object formed by faces $\{f_0, f_1, ..., f_m, f_{d1}, f_{m+1}, ..., f_n, f_{d2}\}$ before and after a single edge collapse (see Fig. 3) to be used

as collapse priority. Note that faces f_{d1} and f_{d2} become singular after the collapse and therefore their contribution to the volume after the collapse is zero.



(a) (b)

Figure 5: Sphere decimation from 480 to 250 triangles. (a) Irregular decimation (b) Regular decimation after priority computation fix

However, a priority computed from this volume gain makes the algorithm to process the shorter edges first, since their smaller adjacent triangles are more likely to introduce a smaller volume change. The areas formed by small triangles are progressively decimated into areas with presumably a bit larger, but still small enough triangles, forcing the algorithm to process them early again. As a result, the decimation runs irregularly around the mesh. Example can be seen in Figure 5a. One can see that the area around the sphere equator remains untouched as the caps of the sphere contain smaller areas with smaller local volumes.

To address this issue, we use global mesh volume gain to compute the priority. The volume gain caused by the edge collapse is computed relatively to the volume of the original mesh, not to the collapse area, i.e. as the sum of volume gains caused by all previous edge collapses and the volume gain caused by the edge collapse[1]. When the area with small triangles is progressively decimated, it grows outwards, the global volume gain increases and the computed priority decreases. Consequently the priority gets lower than the priority in unchanged areas with larger triangles, resulting in regularly performed decimation around the whole mesh (see Fig. 5b) and limiting the undesired variation of spacing between the hull and the original mesh.

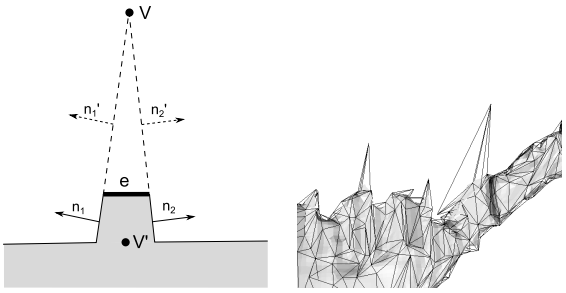
4.2 Algorithm stability

The mesh deformation method, which is described in [4], uses a hull of the mesh to speed up the computation. The vertices of the mesh are linked using barycentric coordinates to the near vertices of the hull, the deformation is then computed on the hull and the results are projected back to the mesh, deforming it accordingly. Because of this, the hull has to preserve the shape of the original mesh closely. Ideally, the spacing between the hull and the mesh is constant and the computed deformation is distributed evenly to the vertices of the original mesh. Note that one vertex of the hull represents a number of vertices of the original mesh and therefore any artifact present on the hull can cause the deformation projection to behave very unexpectedly.

When we used the original algorithm on the surface biomedical data extracted from volumetric data, we observed its high instability, artifacts resembling "spikes" were often present on the produced hull.

If there are two nearly parallel triangles in the edge collapse area (see Figure 6a), the optimal solution to the linear programming problem is a very distant vertex. Provided that the collapsing edge is short, the volume gain caused by the collapse is relatively small, the collapse priority is high and the edge is collapsed, resulting in a "spike" on the surface of the hull (see Figure 6b).

As the number of iterations rises, the artifacts are more common and accumulate. The resulting hull is then unusable for any later use. In order to prevent these artifacts from developing, we added a test that disallows the edge collapses causing them.



(a) (b)

Figure 6: Algorithm instability. (a) Cause of the problem (b) Spikes on the hull

Our test is inspired by Platis and Theoharis [6]. They proposed using Gueziec's [3] test of triangle deviation in progressive hulls to prevent creases on the hull. They test the angle between normals of triangles $\{f_0, f_1, ..., f_m, f_{m+1}, ..., f_n\}$ before and after collapsing the edge they are adjacent to (again, see Fig. 6a). However, as these normals are often unchanged, the spikes may still occur.

We use a slightly different method of checking the decimation quality. We compute angles α_n between normals of triangles $\{f_0, f_1, ..., f_m, f_{m+1}, ..., f_n\}$ adjacent to the vertex V_{new} in pairs. If any angle α_n is larger than the user-specified threshold α_t, we disallow the collapse, because a large angle between the normals of the triangles implies a small angle between the triangles. This small angle between two neighboring triangles indicates an undesired spike on the hull. Figure 7 shows, how are the angles the situation illustrated in Fig. 6a evaluated and since $\alpha_n > \alpha_t$, the collapse is therefore disallowed.

Since our biomedical data mainly consists of smooth surfaces (such as bones and muscles), any local decima-

[1] The collapse of the edge for which the priority is computed

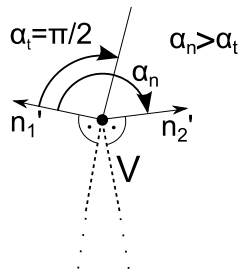


Figure 7: Triangle normal angle check

tion has to be relatively smooth, without any spikes. If some spikes are already present in the original mesh and would result in spike-like area on the hull, they are simply skipped. Experimental results show that these artifacts in the mesh are later "absorbed" by the hull as the surrounding areas are decimated. Therefore, these artifacts do not occur on the hull and the method becomes very stable in the required (and usually large) number of iterations.

4.3 Vertex and triangle quality

Many decimation methods (e.g. [5, 3, 6]) perform vertex valence test. Due to our better priority computation (see Section 4.1), our algorithm creates a regular hull of the mesh. Experimental results show that this test is not needed and the vertex valence is balanced automatically.

The shape of the triangles is also important. Generally, any computations are more stable if performed on compact triangles. Platis and Theoharis [6] suggest performing a triangle compactness test analogically to the test used by Gueziec [3]:

$$c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2}$$

where a is a positive area of the triangle and l_0, l_1 and l_2 are the lengths of its three sides. This number represents a quality of a triangle, smaller for "sliver" triangles that cause numerical instability in later use. Therefore, disallowing the edge collapse if the triangle compactness of any of the faces $\{f_0, f_1, ..., f_m, f_{m+1}, ..., f_n\}$ is lower than an user-specified threshold results in a hull, which contains only sufficient quality triangles.

We use a different simple test. We check the largest inner angle of each triangle (based on the largest dot product of two of the three triangle sides' vectors). If this angle is larger than the user-specified value, the triangle is considered to be narrow, and therefore undesired. If such an angle is present already in the original mesh, we check, if it gets smaller by the planned edge collapse. If the outcome of this test is negative, naturally, we disallow the collapse. This method is slightly more simple to implement and does not require additional triangle area computation, while the results are sufficiently good enough. On the other hand, if the mesh contains many narrow triangles, the algorithm disallows most of the edge collapses and desired number if primitives in the hull is not reached.

4.4 Self-intersection

Sander et al. [7] hypothesize that self-intersection prevention may be unnecessary. In our tests we did not observe any self-intersections caused by our algorithm.

Nevertheless, a self-intersection in the hull is caused by introducing a sharp crease in the mesh. This type of self-intersection may be introduced by possible local imperfections in our data. Figure 8 illustrates an example of this situation. One can see that by adjusting the α_t parameter (see Section 4.2), we can define what constitutes a sharp crease and therefore prevent the possible self-intersection.

This adjustment however globally affects the whole decimation process. Using a very small α_t parameter forces the algorithm to limit the degree of decimation in rough and creased mesh areas. This behavior can be desired (preserving subtle details in the mesh), but results in an uneven mesh decimation.

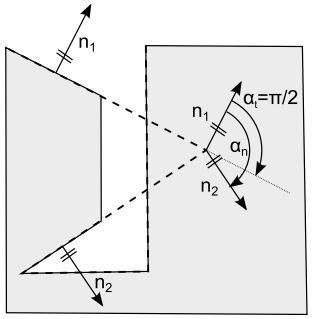


Figure 8: Possible introduction of hull self-intersection.

The self-intersection can also be caused by the very shape of the original mesh, where two mesh surfaces, even without any creases, are close together, resulting in a self-intersection in the hull. In this case, additional set of constrains defined by triangles that could be intersected by the edge collapse has to be added to the linear programming problem, as described in paper [7]. Our algorithm does not prevent this type of self-intersections, since our biomedical data consists mainly of convex and smooth surfaces.

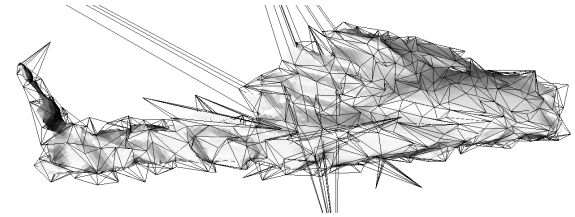
5 Experiments and Results

The method described above was implemented in C++ (MS Visual Studio 2010) and integrated into the MuscleWrapping application[2], which is a part of LHPBuilder software being developed within the VPHOP project [1]. The algorithm was tested on data sets included with this software. The software uses the Multimod Application Framework (MAF) [9], a rapid development visualization system mainly based on Visualization Toolkit (VTK) [8]. For our experiments, a Dell Precision 470 desktop computer (2x Intel Xeon 3.4 GHz, 2 GB DDR2 400MHz RAM, 2x HDD 137 GB SCSI with 10,000rpm, Windows XP Pro) was used.

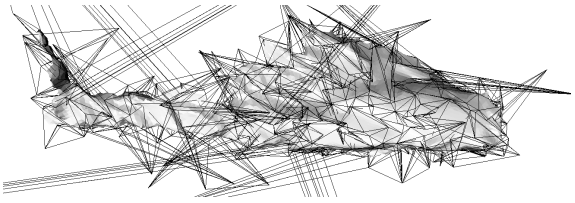
[2] http://graphics.zcu.cz/Projects/Muskuloskeletal-Modeling

5.1 Hull quality and spike prevention

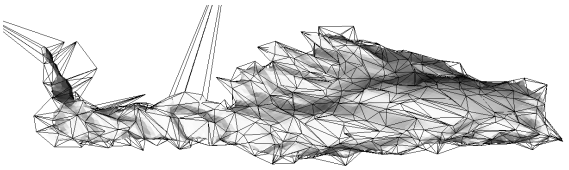
In this section we visually compare the stability of our algorithm to the stability of algorithms based on the *Progressive Hull* method described in this paper. The spike artifacts are commonly present on the hulls created by the original method [7] (see Figure 9a) and the method with a fast priority computation described in Section 3 (see Figure 9b). Hulls created by our implementation of these methods are obviously unusable for later use. Results of the method with a triangle deviation test used by Platis et al. [6] (see Section 4.2, Figure 9c) are slightly better, but the spikes are still present on hull. As we prevent the forming of the spikes specifically, the hull constructed by our algorithm does not contain any (see Figure 9d) and therefore is safely usable for later computations.



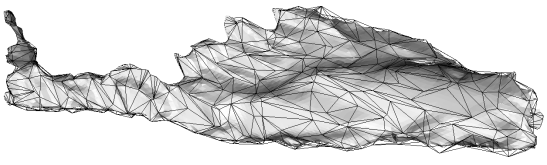
(a) Sander et al



(b) Faster priority computation



(c) Platis et al



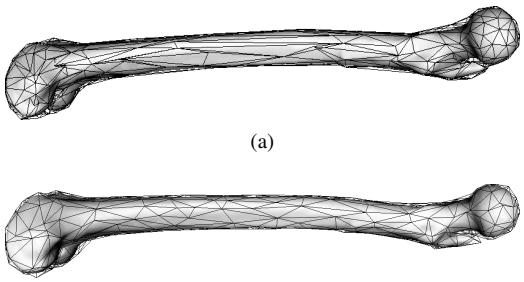
(d) Our method

Figure 9: Comparison of hull quality across the different *Progressive Hull* based methods

The tests shown in Figure 9 were performed on a biomedical mesh of the left *Piriformis* (again extracted from volumetric muscle data) that contains 15000 triangles. The target mesh decimation was set to 90% (90% of the primitives removed), the resulting hull in all four tests contained 1496 triangles.

5.2 Triangle quality improvement

Using the method described in Section 4.3, we achieved a reduction of narrow and sliver triangles on the hull. Visual example can be seen in Figure (see Fig. 10). One can observe a significant triangle shape quality increase.



(a)

(b)

Figure 10: Comparison of triangle quality (a) before triangle angle check (b) after triangle angle check

To confirm this observation, we analyzed the mesh and computed each triangle's compactness using the formula by Gueziec [3] described in Section 4.3. Histograms in Figure 11 show that the most of the low compactness and therefore low quality triangles are removed from the hull.

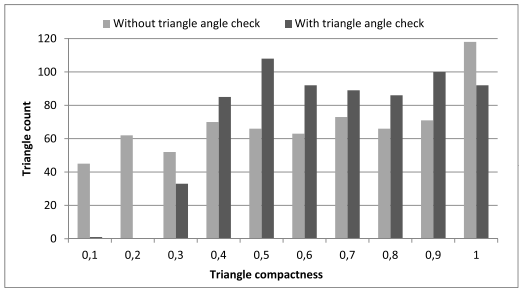


Figure 11: Triangle compactness histograms

In both tests, 90% target decimation was used. The *Femur* mesh was decimated from 13946 to 697 triangles.

5.3 Time consumption

In this section we compare the time consumption of our implementation of the original method [7], the modified method using faster priority computation [6] and our method. We performed tests on 3 meshes with the same setting of 90% decimation.

The results in Table 1 show that our method is approximately nine times faster than the original method by Sander et al. and it may be a couple of seconds slower than the method by Platis et al. [6], however, the additional time consumption is an acceptable trade-off for high quality hulls. Furthermore, considering that the method is typically used in pre-processing, we came to a conclusion that the slowdown of our algorithm in comparison with Platis et al. [6] is insignificant.

Mesh	Method execution time [s]		
	Sander et al.	Platis et al.	Our method
Gluteus Minimus Mesh: 7980 △ Hull: 776 △	104,3	8,6	14,7
Femur Mesh: 13946 △ Hull: 1384 △	215,6	17,9	19,9
Left Piriformis Mesh: 15000 △ Hull: 1496 △	193,4	28,6	23,6

Table 1: Method execution time comparison

5.4 Execution time dependency on the target decimation level

Figure 12 shows the execution time of the algorithm as a variable of a target decimation level. This test was performed on high polygon mesh representation of a *Pelvis* bone. The decimation level denotes how many primitives were removed from the mesh, for example 10% decimation level means that $\frac{1}{10}$ of the original primitive count is removed from the mesh. For this test and default settings, the maximum decimation level was 99.4% (1130 triangles). Note that the preparation time is constant as the measurements were performed using the same mesh and therefore the number of edges entering the priority queue is constant.

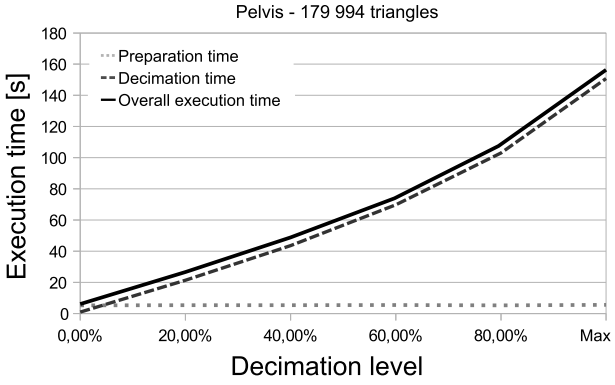


Figure 12: Execution time dependency on target decimation level

In the preparation phase, the priority queue is constructed. Since the priority queue is implemented using a heap data structure, the queue construction is equivalent to a sorting problem and therefore the complexity of the preparation phase is $O(C_p * N * log(N))$, where N denotes a number of edges in the mesh and C_p denotes a time needed to compute the priority.

The decimation itself performs edge collapse on every edge in the queue. The number of the edges in the queue decreases by three with every successful edge collapse. One edge is removed for the collapse itself and two are removed during the area reconstruction process (edges of the triangles, that become singular by this operation, see Section 3). The priorities of the affected edges are updated after every collapse, and the heap property is restored. Since we use a data structure that maintains the neighbors of every primitive in the mesh, the edges that need to be updated in the queue are found in constant time. If we presume that the number of primitives in collapse area is constant, the overall decimation phase complexity is $O(C_c * C_p * \frac{N}{3} * log(\frac{N}{3}))$, where N denotes a number of edges in the mesh, C_p denotes a time needed to compute the priority and C_c denotes a time needed to collapse the edge. Note that $C_c \gg C_p$, as collapsing the edge requires a linear programming problem solution.

5.5 Additional hull results

In figures 13-17 we present the results of our algorithm on several other biomedical meshes.

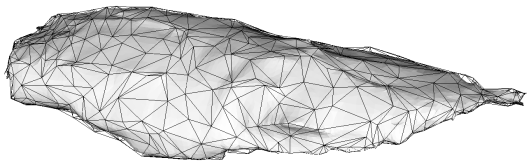


Figure 13: Vastus Lateralis, 19970 △; 1980 △ in the hull

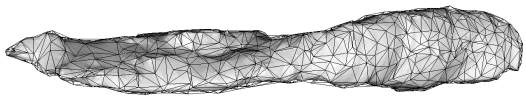


Figure 14: Vastus Medialis, 19986 △; 1982 △ in the hull

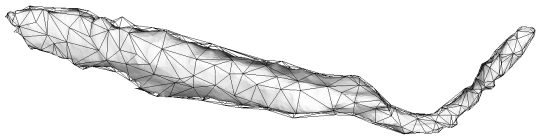


Figure 15: Psoas, 9988 △; 984 △ in the hull

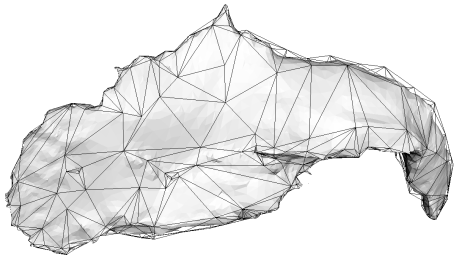


Figure 16: Iliacus, 9968 △; 964 △ in the hull

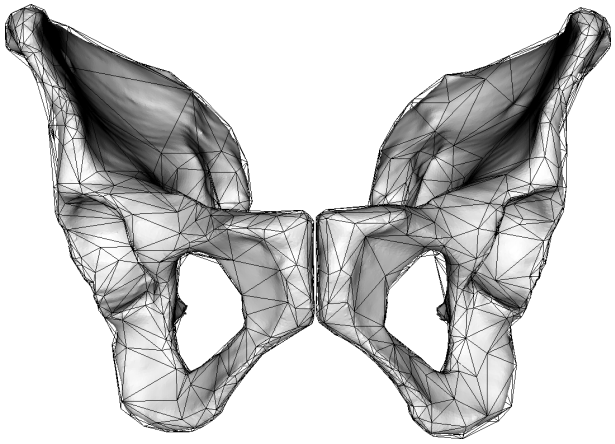


Figure 17: Pelvis, 179994 △; 1130 △ in the hull

6 Conclusion and future work

The proposed method constructs coarse outer hulls that do not contain artifacts that may appear when using the original *Progressive Hull* method [7]. It also offers a better overall quality results. Our algorithm was successfully used in a mesh deformation technique [4] and allowed it to be more precise, due to the fact that the *Progressive Hull* shape preservation is very high and consistent.

Time consumption of the algorithm is a problem that remains unsolved. The method, though it is nine times faster than the original one (see Section 5.3), is still time consuming. For many applications (including ours) this is not a serious drawback, since the method runs only once in the pre-processing.

The future and current work includes usage of a graphics processing unit (GPU) for a faster hull construction as well as further triangle quality enhancement using a better triangle compactness test in order to allow the algorithm to always perform enough iterations to create a coarse outer hull, that contains the desired number of primitives.

7 Acknowledgment

This work was supported by the Information Society Technologies Programme of the European Commission under the project VPHOP (FP7 ICT-223865) and by the Ministry of Education of The Czech Republic under the project 7E11016.

References

[1] VPHOP: The osteoporotic virtual physiological human. http://www.vphop.eu/.

[2] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shape. *ACM Transactions on Graphics*, volume 13(1):pp 43–72, 1994.

[3] André Guéziec. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, volume 5:pp 168–189, April 1999. Chapter 5: Description of the Algorithm.

[4] Josef Kohout, Petr Kellnhofer, and Saulo Martelli. Fast deformation for modelling of musculoskeletal system. In *Proceedings of the International Conference on Computer Graphics Theory and Applications: GRAPP 2012*, pages 16–25, Rome, February 2012.

[5] Peter Lindstrom and Greg Turk. Fast and Memory Efficient Polygonal Simplification. *Visualization Conference, IEEE*, pages 279+, 1998.

[6] Nikos Platis and Theoharis Theoharis. Progressive hulls for intersection applications. *Comput. Graph. Forum*, volume 22(2):pp 107–116, 2003.

[7] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 328–329, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[8] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit, Third Edition*. Kitware Inc., August 2004. ISBN: 1-930934-12-2.

[9] Marco Viceconti, Luca Astolfi, Alberto Leardini, Silvano Imboden, Marco Petrone, Paolo Quadrani, Fulvia Taddei, Debora Testi, and Cinzia Zannoni. The multimod application framework. In *Proceedings of the Information Visualisation, Eighth International Conference*, pages 15–20, Washington, DC, USA, 2004. IEEE Computer Society.

Ray-casting point-in-polyhedron test

Denis Horvat*
Supervised by: Borut Žalik†

University of Maribor
Faculty of Electrical Engineering and Computer Science
Laboratory for Geometric Modelling and Multimedia Algorithms
Smetanova ulica 17, SI-2000 Maribor, Slovenia.

Abstract

This paper considers a ray-casting point-in polyhedron test. Although it is conceptually the simple extension of a well-known point-in-polygon ray-casting algorithm, various practical problems appear in 3D, especially, when the boundary of a geometric object is represented as a triangulated surface. When a larger number of points have to be tested regarding their positions on the considered geometric object, preprocessing may drastically reduce the testing time. This paper considers comparisons between three such methods: The first uses a k-dimensional tree (kd-tree), the second an octree, and the last is based on a three-dimensional uniform grid (3D grid). The core operation for all three methods is the ray-casting, by which the odd-even rule can be efficiently applied. Ray-casting can be susceptible to the rounding errors, which are also considered in this paper.

Keywords: 3D inclusion test, spatial subdivision, kd-tree, 3D uniform grid, octree

1 Introduction

Knowing whether a certain point lies inside a given polyhedron can be beneficial within a wide-range of computer graphic applications. This test is often called the inclusion test and is usually used as a base operation in conjunction with more complex ones, so it is essential that it satisfies certain criteria. These usually involve speed, robustness, and memory usage. The inclusion test has its well-known usages in the field of collision detection, where it helps to ensure that objects do not fall through the ground or go through walls. It can also be found in physics simulation and artificial intelligence. Many efficient methods regarding the inclusion test were presented [15, 7, 3, 1] that all have their advantages and disadvantages, depending on the tested object. They can be divided into two basic groups: methods that require a data preprocessing phase, and those that do not. The latter includes ray-crossing methods [4, 3], the angular method [7], barycentric coordinates [1], the winding number method [7], and others. The time-complexity for a single tested point in those methods without preprocessing is *O(n)* [15, 10], with *n* being the number of vertices. Whilst these methods are suitable for small a number of tested points, they become less and less appropriate when the number of tested points increases. At that stage it might be better to consider methods that perform data preprocessing before executing the actual inclusion test. During the preprocessing phase, data is systematically organized, and is later used as input for the inclusion test. Data preprocessing is usually the most intensive operation, but is only performed once. Many structures can be used for data preprocessing, such as uniform grids [15, 12]. The expected time complexity of a inclusion test for methods that use data preprocessing is *O(log(n))* or even *O(1)*. In this paper, the advantages and drawbacks of three data structures that can be used for the inclusion test are analized, described, and compared with each other. The problem of inclusion is solved for the boundary representation (B-rep) of a polyhedra that consist of triangular meshes.

The paper is organized in 7 sections. Section 2 briefly describes the related works that have already successfully solved the problem of inclusion. Section 3 describes how to find an intersection between a point and triangular plane. Section 4 describes how to subdivide space using a kd-tree and an octree. Solutions for how to traverse the mentioned tree structures are also given. Section 5 explains how to voxelise a scene and use it for the inclusion test. Section 6 tackles those problems that may arise from rounding errors when using rays. Section 7 describes and compares the results of experiments conducted on a single workstation using the preprocessing methods described in sections 4 and 5. Section 8 summarises the paper.

2 Related work

Very few original methods seem to have been developed for the inclusion test in three-dimensional space. Most of them are just extensions of their two-dimensional counterparts. Feito and Torres [2] solved the problem of inclusion

*denis.horvat@uni-mb.si
†zalik@uni-mb.si

without the usage of trigonometric functions, and without solving any equations. In [16], a layer-based structure was used for scene preprocessing, where the occlusion relation between faces and edges were calculated, and the faces were then projected on sequentially arranged layers. Later, a binary search algorithm was used on the preprocessing data. The storage-space used during preprocessing was greatly reduced because much of the information for the polyhedrons is represented implicitly. Other various methods can be used by algorithms that are popular in computer graphics. The inclusion test using rays is closely related to ray-tracing, as each ray must be processed by a ray tracing-algorithm in order to determine the intersection points (if they exist) with the tested object. In his PhD thesis [6], V. Havran conducted a comprehensive comparison between twelve commonly used ray-tracing algorithms for a set of thirty test scenes. His findings were, that ray-tracing algorithms based on a kd-tree achieved the best results in comparison with other tested algorithms. Octrees were placed second.

3 Inclusion test using ray-casting

Methods that involve ray-casting follow a simple principle when it comes to the inclusion tests. The basic primitive in ray-casting is a ray $\overrightarrow{R}$, which is defined by its origin O and a normalized direction vector $\overrightarrow{D}$. The ray is cast from a tested point p, that serves as its origin, in any direction. Next, the number of intersections is determined between the polyhedron and the ray. If this is an odd number, p lies inside the polyhedron, otherwise it is outside (odd-even rule). Each individual triangle that is a part of triangular mesh is tested with the casted-ray in order to count the number of intersections for a single point p. This is done by first finding the intersection of a ray with a triangular plane. Any point P lying on $\overrightarrow{R}$ can be defined by a parametric representation (1) of that ray, where t is the signed distance.

$$P = O + t\overrightarrow{D} \tag{1}$$

The signed distance t can be calculated from (2), where d is the distance of the plane from the origin and $\overrightarrow{N}$ the plane normal vector.

$$t = \frac{-(O\overrightarrow{N} + d)}{\overrightarrow{D}\overrightarrow{N}} \tag{2}$$

If $t < 0$, then the triangular plane lies behind the vector origin any tests for that particular triangle can be aborted (figure 1a). This also happens when the ray and triangular plane are parallel to each other (figure 1b), or in geometry terms, when a dot product between the triangular plane and $\overrightarrow{D}$ equals zero. In the case of a positive t, intersection P with a triangular plane is calculated (figure 1c) using (1). This intersection point is then tested with one of the point in polygon tests without preprocessing mentioned in section 1. As already stated, this part can be susceptible to rounding errors, which are addressed in section 6.

Thus, the basic inclusion test is already possible, but every triangle is tested for each point, which leads to undesirable results regarding speed. Consequently, data preprocessing is introduced to ensure that the minimal number of triangles is tested. There are many structures that can be used for spatial subdivision that all have their advantages and disadvantages depending on the given scene [6, 1]. The next two sections explore three of these structures: kd-tree, octree, and uniform grid.

It is also important to mention, that in order for the described methods to provide valid results, the polyhedra should not contain missing triangles or cracks, as the ray could go through that hole and consequentially the point would be classified incorrectly. This problem can be tackled by testing each point using more rays, and selecting the result with the majority.

4 Using tree structures for spatial subdivision

One of the ways to minimise the number of tested triangles for each individual ray, is to recursively subdivide the space by constructing a tree structure. The divided space volume is presented in the form of axis-aligned bounding boxes (AABB). Each node is associated with his AABB, whiles the bounding box of the root node covers the whole of scene S. Nodes that have no child nodes are called leaves. Leaves that contain at least one object of S are called full leaves, otherwise they are empty leaves.

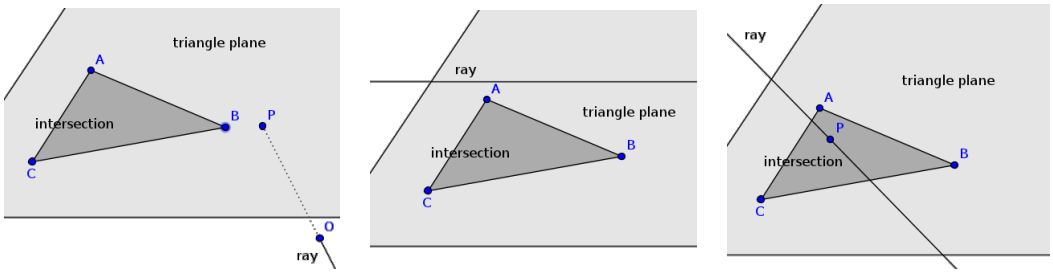
4.1 Kd-trees

Space can be subdivided by a k dimensional tree (kd-tree). Here, the number of dimensions k is limited to three. A kd-tree is a binary tree, which recursively divides space into two new AABB. The division stops after a given criteria is reached. What makes the kd-tree unique is that AABB is divided by a splitting plane, which can be positioned anywhere as long it is perpendicular to its dividing axis. One of the ways to choose the dividing axis is to change it in the cyclic order x,y,z one axis per each depth, usually starting with x. The method of always splitting the longest axis can also be considered. An example of a simple subdivision within a two-dimensional space using a kd-tree, is shown in Figure 2.

At this stage, two important questions regarding the tree construction need to be answered [6]:

- Where to position the splitting plane?
- When to stop dividing?

The answer regarding the first question is very important as it can improve the overall speed of the tree-traversing step. Several methods are known for the positioning of the splitting plane:



(a) Plane lies behind ray origin

(b) Ray and plane are parallel to eachother

(c) Ray intersects plane in P

Figure 1: Different cases of ray-plane intersection

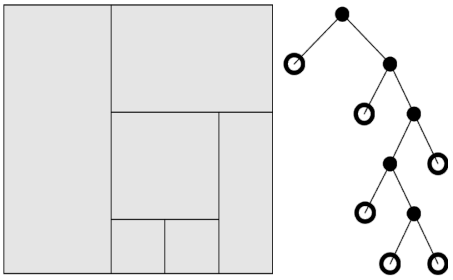


Figure 2: Example of a non-balanced kd tree

- **Spatial Median:** Existing AABB is always divided into two halves by its splitting plane.
- **Object Median:** The splitting plane is positioned in such a way, that the number of objects on each side is roughly the same. This is generally a bad idea [6], because no leaf in the kd-tree will be an empty leaf, meaning that when the tree is traversed, all the objects would need to be checked for each leaf the ray passes through.
- **Cost Model:** This method determines the optimal splitting plane position by calculating the splitting cost using a heuristic for each considered plane. The heuristics used is called *the surface area heuristic or SAH*, as described in [6, 11]. *SAH* uses the idea that the chance of a ray hitting an AABB is proportional with its surface area. This means that it is best to isolate the bigger empty nodes, so that the ray has the highest chance of passing through them unhindered. The cost is calculated using (3), where C_t is the cost of traversal, p_l, p_r the probability of a ray intersecting the left or right node, and C_l, C_r the estimated cost of the left and right sub-node.

$$C_n = C_t + p_l \cdot C_l + p_r \cdot C_r \tag{3}$$

Next, a termination criteria is determined for when to stop dividing and classifying the current node as a leaf. The criteria used is called *ad hoc* termination criteria, where the current node C_n becomes a leaf when a tree depth reaches a certain threshold T_{max}, or the number of objects in C_n is less than the constant O_{min}. Both constants are determined by the user.

After space has been partitioned using a kd-tree, the data obtained can be used when an inclusion test is performed. Only those leaves that the ray intersects are examined instead of the whole scene. This is done with the tree traversal. The recursive ray traversal algorithm TA^A_{rec} [6, 5] was used to traverse the kd-tree. Intersections with a polyhedra are determined by testing all the objects from intersected leaves using the method described in Section 3. The algorithm was published by [8] and uses near-far node classification based on the ray's origin. When a node is traversed, it is calculated which one of his child nodes must be traversed, and which can be skipped. Three cases of traversal are possible: traverse near, traverse far, traverse near and then far. The algorithm uses a stack structure to keep track of the nodes that need to be traversed. The result of the traversal can be seen in Figure 3.

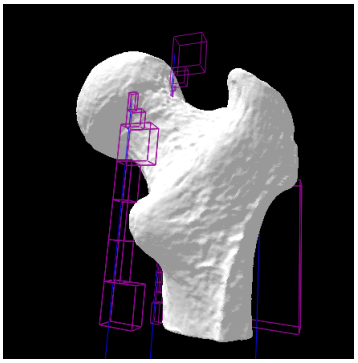


Figure 3: Result after kd-tree traversal using $T_{max} = 21$ and $O_{min} = 4$ and SAH split division criteria.

4.2 Octrees

Octrees are usually used to partition three dimensional space by dividing it into eight octants. Another difference

compared with the kd-tree is that all three planes are divided per each depth, instead of just one. Each node has its own AABB and the leaf nodes have a list of objects that are contained within that AABB. Triangle-cube intersection algorithm [9] was used to classify objects to their individual AABB. The space is subdivided until the *ad-hoc* termination criteria described in subsection 4.1 is reached. Simple subdivision of the three-dimensional space using octree, can be seen in Figure 4.

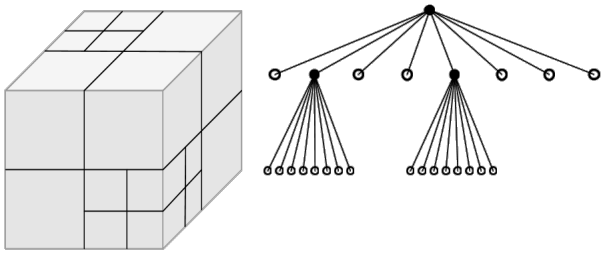


Figure 4: Example of an non-balanced octree.

The traversal step is performed by using an *efficient parametric algorithm for octree traversal* as published in [13]. Similarly to section 3, this algorithm leans on the fact that any point lying on ray can be defined by a parametric representation (1) of that ray $\overrightarrow{R}$, where t is the signed distance. Let b_{max} be the AABB maximum boundary point and b_{min} its minimum. If at least one positive distance t exists on $\overrightarrow{R}$ that is between the AABB boundaries, then $\overrightarrow{R}$ intersects that AABB. The algorithm works with distances from the ray origin O to the nodes limits b_{max} and b_{min}. These distances t_{max_i} and t_{min_i} are calculated using the equations (4), where $\overrightarrow{D}$ is the ray orientation and i one of the coordinate.

$$t_{max_i} = \frac{b_{max_i} - O_i}{\overrightarrow{D_i}} \qquad t_{min_i} = \frac{b_{min_i} - O_i}{\overrightarrow{D_i}} \tag{4}$$

This calculation is only done for the root node. Distances for child nodes are incrementally calculated from parent nodes using three additions and three shifts as the algorithm recursively progresses. For each voxel, the first crossed node is determined (if it exists). Based on the first node, the next nodes can be found until ray exists the current voxel. The algorithm recursively progresses until the leaf nodes are reached. The result can be seen in Figure 5.

5 Uniform grids

For the inclusion test using uniform grids, the algorithm from [12] was implemented. The mentioned algorithm is a three-dimensional extension of the *two-dimensional cell-based containment algorithm (CBCA)* [15]. During

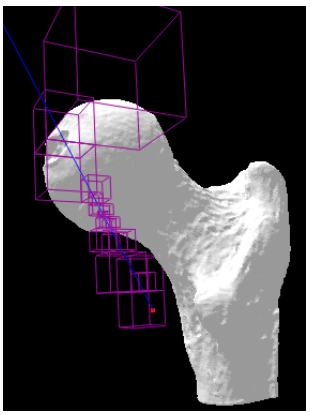


Figure 5: Result of octree traversal for one ray using T_{max} = 20 and O_{min} = 50

the preprocessing phase, CBCA constructs a raster and places it onto a given scene. Each cell is marked as: inside, outside, or as a border cell. Flood fill algorithm is used in order to determine whether a cell is located inside of a polygon. When the actual inclusion test is performed, the algorithm calculates in which cell the tested point is located, and then checks the cell's status. In the case of a cell being marked as inside, the point is declared to be inside, otherwise it is outside. When it is marked as a border cell, additional testing is performed, depending on whether a detailed inclusion test is requested.

In a three-dimensional space, a grid of uniform voxels is used instead of a raster, and ray casting is performed instead of the flood fill algorithm. The object is voxelised, but similarly to CBCA, not all the cells (here voxels) lie completely inside or outside. If the approximation test suffices, then the voxels are marked as inside if 50% of their volume lies inside the tested object, and vice versa. When a detailed test is required, then those voxels that contain the object surface are marked with one more additional bit. Additional tests are performed for those points that lie inside such voxels.

5.1 Voxelisation

Voxelisation is done by an algorithm described in [14], but the idea for uniform voxelisation of three-dimensional polygonal objects or polyhedra comes from [17]. So-called optimised ray-casting is used to determine whether a voxel lies inside or outside of a given object. Voxelisation is performed in two steps:

- In the first step, the initial AABB of the scene is calculated and its xy plane is partitioned using a quadtree. Partitioning stops when the *ad hoc* termination criteria described in section 4.1 is reached. Each leaf in the quadtree corresponds to an AABB with its deph equal to the initial AABB. Each leaf keeps track of all objects that are contained within its AABB. A

Cohen-Sutherland line clipping algorithm [3] is used for determining the containment of an object to the specific AABB. Once the space is partitioned and the tree is built, the next step can commence.

- The second step begins by covering the scene with a uniform grid, containing $m \times n \times p$ voxels. The question regarding the grid resolution must be answered, as it influences the preprocessing speed, memory usage, and accuracy. In 2D, an equation can be used to determine the grid's size [15]. Here, three methods are considered [12]: The first method has a fixed voxel size that is applied to all scenes. The voxel size can also be determined by the user, which gives him/her more control over each individual scene. The third method calculates the voxel size in relation to the scene's extent. After the grid has been constructed, $m \times n$ rays are cast parallel to the positive z axis. One of the $m \times n$ voxel's center then serves as the origin for each ray. The z value of each individual origin is equal to the minimal z value of the initial AABB. For each individual ray, an AABB is found by recursively looking into the quadtree, obtained during the previous step, and finding the leaf that corresponds to the x, y coordinate of the ray's origin. All objects in this AABB are checked for intersections and for each intersection found, its z coordinate value is stored inside a linked list. A new list is always generated for each ray. This list is then sorted in ascending order. For each voxel that is traversed by a ray, the number of values in the list are counted that are smaller than the current voxel center z value. The odd-even rule is applied, which means that if the number of values is odd, then the voxel lies inside of the object, or outside if it is even. The result can be seen in Figure 6.

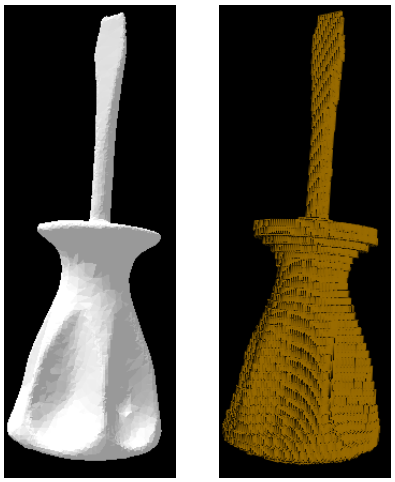


Figure 6: Voxelisation using 128^3 voxels.

5.2 Inclusion test

After the object has been fully voxelised and state of each voxel is known, inclusion test is as simple as reading the voxel's state in which the tested point is contained. The coordinates of a voxel v(m_v, n_v, p_v) for a tested point p(x_p, y_p, z_p) can simply be calculated [12] using the following equations (5):

$$m_v = \frac{x_p - x_{min}}{size_m} \quad n_v = \frac{y_p - y_{min}}{size_n} \quad p_v = \frac{z_p - z_{min}}{size_p} \tag{5}$$

Problems occurs when the objects' boundaries are passing through the voxel. Part of the voxel may be located inside and the other part outside the object, but the voxel can only have one of the two states. This can cause that the result of the inclusion test is incorrect. When the voxel is marked as a boundary, a ray parallel to its z axis is cast from the tested point. The ray stops after a non-boundary voxel is encountered. Intersections are determined (as described in subsection 5.1) and if their number is odd, the status of the tested point is opposite to that of the voxel status in which it is contained.

6 Numeric stability

Sadly computer arithmetic is finite. This causes rounding errors to occur and consequently, inclusion tests can return false results.

As shown in section 3, the intersection between a ray and triangular plane is calculated. This intersection point is then tested using one of the basic point-in-triangle tests and if it is located inside, then intersection occurs. The biggest occurs when this intersection point is located just on the triangle's edge (or very near). Rounding errors can cause the calculated point to be slightly shifted and can now be falsely located in one of the neighbouring triangles. This means, no intersection with that triangle will be found and the result from the inclusion test will be incorrect because the methods used rely on the fact that number of intersections is calculated correctly, so that the odd-even rule can be applied. Robustness was achieved through shared calculations [1] between triangles that share a common edge, using the triple scalar product. All the floating point numbers were compared using relative tolerance comparison. It is important to say, that although the mentioned test improves the robustness when checking for intersections, it is still not 100% accurate.

7 Results

The solutions were tested and compared on a desktop workstation running on Windows 7 using the following

hardware: an Intel i5 processor with a clock speed of 3300 MHz and 4 GB (DDR3) of RAM. All the algorithms were implemented in C++ using the Qt framework. The OpenGL graphical library was used for visual presentation. The structure for storing the model data consisted of a list of triangles and vertices. Normal vectors for each of the triangles were precalculated, when the model was loaded into the memory. Data about the models used for testing, are shown in Table 1 and their thumbnails can be seen in Figure 10. For each result, the average from three measurements was used.

Table 1: Model data

model:	num. vertices	num. triangles
pumpkin	5002	10000
cat	7335	14634
owl	19884	39764
bust	47516	95028
gren1	122227	134016
isis	93823	187642
tiger	309403	618786
gren2	957507	1072128

Total of three methods were implemented for the point-in-triangle test: the angle sum method, the winding number method and a method that uses barycentric coordinates. In the presented tests the method using barycentric coordinates came out on top as it was about 155% quicker compared to the winding number method and more than 223% than the angle sum method. Consequently, the method using barycentric coordinates was used in all the future tests.

Table 2 shows the results for each method implemented. The inclusion test was executed on 300,000 points that were randomly placed on the scene. For the kd-tree (Kd HS represents a tree built using the object median criteria, whiles kd SAH uses the surface area heuristic) and the octree, *ad-hoc* termination criteria was used where $T_{max} = 16$ and $O_{min} = 6$. The object was voxelised using a grid size of 256^3 voxels for approximate testing, where the results were about 99.3% accurate. An accurate grid test was not used for comparison because classification of the boundary voxels took too long. If antialiasing is used during voxelisation, as described in [14], some voxels can still be missed and are not classified as boundaries. Consequently, each voxel must be tested using the *triangle-cube intersection algorithm* [9] for all triangles that are contained within the quadtree node that correspond to the current ray. As soon as only one triangle is found, the boundary test for that voxel can be aborted and the voxel classified as boundary. For example, the preprocessing for the model *cat* took more than 2.5 minutes, which is not even remotely comparable with other methods.

The preprocessing phase T_p and the executing phase T_e were measured for each method. The results for preprocessing T_p are shown in Figure 7. Time required for preprocessing rises with number of triangles on the scene. Preprocessing for kd-trees executes faster than for octrees. This happens because the algorithm for octree must classify each individual triangle to its octant by using the *triangle-cube intersection* test, for each triangle. Classification for a kd-tree only consists of a test that simply determines which side of the splitting plane the triangle is located.

Table 2: CPU(s) for times used for preprocessing and executing for all the tested models, using the implemented methods.

model	Kd HS		Kd SAH		Octree		Grid AP	
	T_p	T_e	T_p	T_e	T_p	T_e	T_p	T_e
pumpkin	0.02	2.19	0.12	2.01	0.12	3.18	2.29	0.05
cat	0.05	1.76	0.12	1.51	0.19	2.2	2.28	0.03
owl	0.06	2.90	0.28	2.70	0.45	4.01	4.07	0.06
gren1	0.08	5.26	0.70	1.95	2.72	2.26	8.85	0.05
bust	0.08	4.32	0.53	3.33	1.12	4.15	7.27	0.03
isis	0.13	7.98	1.33	5.28	2.46	5.51	14.6	0.05
tiger	0.20	11.1	2.95	4.92	10.9	4.31	36.7	0.05
gren2	0.29	21.7	4.49	11.85	20.3	2.84	62.3	0.05

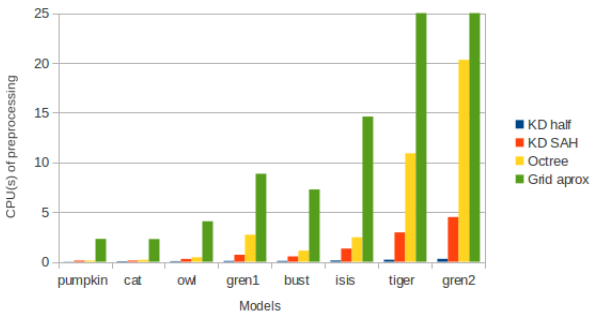


Figure 7: T_p for each model and using all the implemented methods

The results featured in Figure 8 show the execution times T_e after the preprocessing has already been done. The expected execution time for three-dimensional grids is *O(1)* so it is constant and executed the fastest, but the result is only a approximation. The kd-tree build with the SAH heuristic performed best in almost all tested cases when the results were expected to be accurate. It was outperformed by octree in *gren2* where the scene was heavily divided.

Figure 9 shows the results, where each column represents the sum of T_p and T_e for each method implemented on the tested scenes. The kd-tree using the SAH heuristic *KD SAH* perfomed the best in all scenes. The kd-tree build with the half split criteria *KD HS* performed better than the octree in almost all cases except on the *gren1* scene where it was outperformed by a small margin. Even if octree traversal is sometimes done even faster than the traversal of kd-trees, the preprocessing phase is always slower and

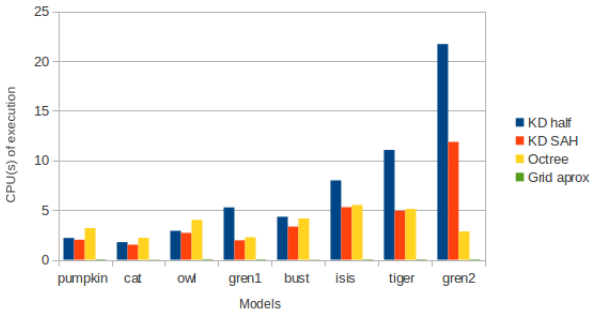


Figure 8: T_e for each model and using all the implemented methods

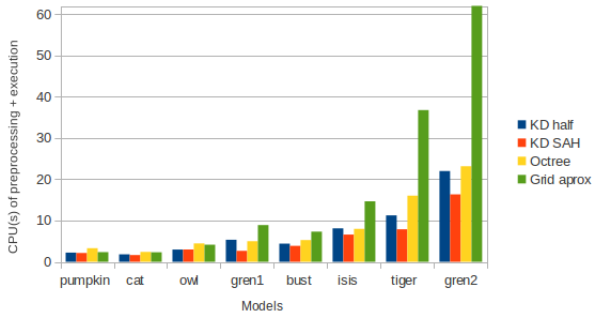


Figure 9: $T_p + T_e$ for each model using all the implemented methods

that is why the total performance of kd-trees is usually better.

8 Conclusion

In this paper, three different methods were used for solving the problem of inclusion using spatial subdivision as preprocessing. Our conclusions support the ones of [6], namely that for a small number of rays cast, data preprocessing does not pay off. After testing the mentioned scenes, the kd-tree using the SAH heuristic gave the best results, except for densely occupied scenes where SAH could not be fully utilized. There is still much that can be done regarding preprocessing, such as cutting off empty space in kd-trees or the usage of an octree special variant called octree-R. Therefore no final answer can yet be given regarding the fastest method for the inclusion test within three-dimensional spaces.

References

[1] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufman Publishers, 2005.

[2] F. Feito and J. Torres. Inclusion test for general polyhedra. *Computers and Graphics, Volume 21 Issue 1*, pages 23–30, 1997.

[3] J. Foley, A. van Dam, S. Feiner, and J. Hudgens. *Computer graphics: principles and practice in C (2nd ed.)*. Addison-Wesley Professional, 1996.

[4] M. Gombosi and B. Žalik. Point-in-polygon tests for geometric buffers. *Computers and Geosciences, Volume 31 Issue 10*, pages 1201 – 1212, 2005.

[5] M. Hapala and V. Havran. Review: Kd-tree traversal algorithms for ray tracing. *Computer Graphics Forum, Volume 30 Issue 1*, pages 199–213, 2011.

[6] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University, 2000.

[7] P. Heckbert. *Graphics Gems IV*. Morgan Kaufman Publishers, 1994.

[8] E. Jansen. Data structures for ray tracing. *Data Structures for Raster Graphics*, pages 57–73, 1986.

[9] D. Kirk. *Graphics Gems III*. Morgan Kaufman Publishers, 1994.

[10] J. Li, W. Wang, and E. Wu. Point-in-polygon tests by convex decomposition. *Computers and Graphics, Volume 31, Issue 4*, pages 636–648, 2007.

[11] J. MacDonald and K. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer: International Journal of Computer Graphics, Volume 6 Issue 3*, pages 153–166, 1990.

[12] K. Ooms, P. De Maeyer, and T. Neutens. A 3d inclusion test on large dataset. *Developments in 3D Geo-Information Sciences*, pages 181–199, 2010.

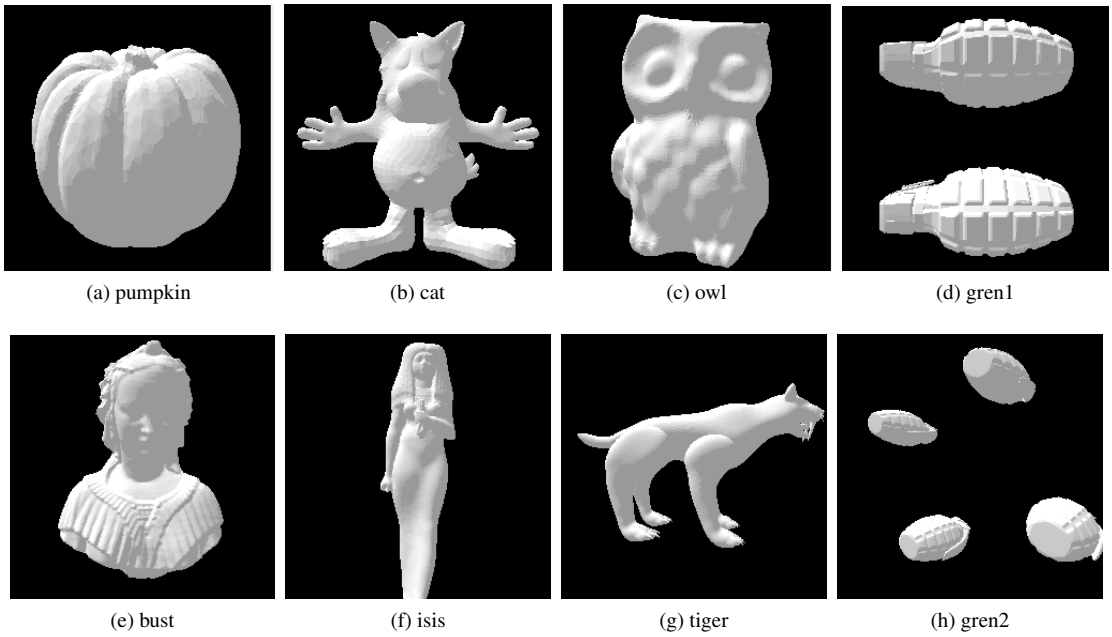
[13] J. Revelles, C. Urea, and M. Lastra. An efficient parametric algorithm for octree traversal. In *WSCG'00*, pages –1–1, 2000.

[14] S. Thon, G. Gesquire, and R. Raffin. A low cost antialiased space filled voxelization of polygonal objects. *GraphiCon 2004 proceedings, Moscou*, pages 71–78, 2004.

[15] B. Žalik and I. Kolingerova. A cell-based point-in-polygon algorithm suitable for large sets of points. *Computers and Geosciences, Volume 27 Issue 10*, pages 1135 – 1145, 2001.

[16] W. Wang, J.Li, H. Sun, and Enhua Wu. Layer-based representation of polyhedrons for point containment tests. *Ieee transactions on visualization and computer graphics, Volume 14, Issue 1*, pages 73 – 83, 2008.

[17] Roni Yagel, Daniel Cohen, and Arie Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications*, 12, 1992.



(a) pumpkin (b) cat (c) owl (d) gren1

(e) bust (f) isis (g) tiger (h) gren2

Figure 10: Thumbnails of the used models

Discovering molecules: Pass planning through a gap

Mgr. Jan Byška*
Supervised by: doc. Ing. Jiří Sochor, CSc.†

Department of Computer Graphics and Design
Faculty of Informatics, Masaryk University
Brno, Czech Republic

Abstract

We present a new algorithm for a molecular pass planning through a circle. Our algorithm can solve the given problem with the significant improvement of accuracy for arbitrary shaped molecules in comparison with the method using a minimal bounding sphere. This accuracy is gained by eliminating the overestimation of the substrate size by the bounding volume approaches. Our approach is particularly beneficial in cases where the bounding volume fits poorly to the substrate geometry as is the case with oblong shaped substrates. We are using a sampling-based version of the motion path planning and the Delaunay triangulation to arrange the substrate for the space search. The successor configurations are then computed incrementally from the already known configurations until we find a connected path of the substrate through a circle or we can claim that such path does not exist.

Keywords: motion path planing, Delaunay triangulation, computational chemistry

1 Introduction

Proteins are irreplaceable parts of every life form. They are involved in many vital processes, for instance they catalyze various chemical reactions, work as antibodies or even transfer signals between distant cells. The detailed analysis of the complex protein structures using methods of computational geometry can significantly help understanding the purpose and chemical principles of specific protein molecules.

One of the most researched properties of the protein molecule is the existence of a protein channel. The channel represents an empty space connecting the inner part of the protein, called *active site*, with its surface. They allow, upon certain conditions, other molecules (*substrates* or *ligands*) to reach the active site, where the reaction between protein amino acids and the substrate can undergo. For biochemists, the knowledge, whether the substrate can enter the protein or not, is important for instance in the process of drug design.

*xbyska@fi.muni.cz
†sochor@fi.muni.cz

As was shown by Petr Medek et al. [3], protein channels can be represented as circular-profile shaped tunnels. The current methods use a minimal bounding sphere (*MBS*) approach to compute whether a substrate can pass through the protein channel. This method is simple because we only need to compute the MBS enclosing all atoms of the substrate and then compare its radius with the narrowest cross-section radius of the channel. However, it is not suitable in cases where the substrate has an oblong or polymer-like shape. In these cases the protein channel has to be significantly bigger to let through the overestimated MBS in comparison with the real scenario, where the oblong substrate can pass through a narrower channel.

The method could be improved by using different types of bounding volumes. The real molecules, however, have often very complicated structures and therefore this solution will not suffice in general cases. We decided to develop a new algorithm based on the motion path planning. Our algorithm can solve the given problem for arbitrary shaped molecules since it is not using any bounding volume but the substrate geometry itself. It computes a path of a set of spheres (which represent atoms of a substrate) through a gap approximated by a circle. Moreover, the algorithm can be generalized to use a cross-section of the arbitrary shape.

As such it can be used in the process of detection whether the ligand can pass through the protein channel. If we make an assumption that these channels consist of large and wide corridors connected by narrow gaps we can then, without loss of generality, anticipate that these wide channel segments are large enough for the substrate to pass through in an arbitrary configuration. Hence, we can use previous inaccurate but fast algorithms for these parts of the channel and focus rather on the narrow holes connecting them. The narrow parts of the protein channel can be, for instance, sampled by a set of circles and our algorithm can handle each circle individually.

The goal of this paper is to present the new algorithm that can detect whether a given substrate can pass through a circle. In the section 3 we briefly remind some of the basic terms of motion planning theory. In the section 4 we focus on the detail description of the algorithm. The evaluation of the complexity of the algorithm is presented in the section 5 and we conclude with the section 6.

2 Previous work

One of the most important task in the computational chemistry is called a channel detection. The approach based on the computational geometry that uses the Voronoi diagram and the Delaunay triangulation to compute a protein channel as a set of spheres, was described by Petr Medek et al. [3]. The connected channel can be created from this set by an interpolation. The problem of a substrate passing through the protein channel can then be solved by comparing the bounding sphere of the substrate with the smallest radius of the channel.

Another method for solving this problem was presented by Haranczyk and Sethian [1]. They sampled the high-dimensional configuration space to find a path through a protein.

There are generally two possible approaches to handle the motion path planning problem. The *combinatorial approach* [2] or the *sampling-based approach* [2]. We will describe them in detail in the next section, after the necessary background is given. The motion path planning theory was initially developed for the navigation of the robot through the complex environment. In molecular chemistry, however, several major problems were successfully solved by motion planning approaches, such as *protein folding* or *ligand docking*.

The protein folding is the process when a polypeptide is folded and connected into a final protein structure. It was solved by the motion planning by Song and Amato [5]. The second problem – the ligand docking – refers to the issue of a substrate inserting itself into a protein cavity while satisfying other constraints, such as maintaining the low energy. An algorithm using the probabilistic motion planning to compute the most energetically favorable path between any initial and goal ligand shape was presented by Singh et al. [4].

3 Motion planning concept

Since the motion planning theory was initially developed for the navigation of a robot through a complex environment, it is using terms such as a *robot* or a *path*. In this section we briefly remind some of this basic terms and show how to adapt them for a substrate passing through a protein channel (for more information about motion planning see LaValle 2006 [2]).

The basic element of the motion planning theory is a *state*. Each state represents a specific transformation that can by applied to the robot and thus defines a specific position and orientation of the robot in a *world space* $\mathcal{W}$. In our case, the world space is three-dimensional and the robot $\mathcal{R} \subset \mathcal{W}$ represents the substrate moving in it. Therefore the state representing the substrate will be actually a six-dimensional vector $\vec{q}(x_t, y_t, z_t, u_t, v_t, w_t)$ with three positional and three rotational coordinates.

The set of all states is called the *configuration space* and it is often denoted as $\mathcal{C}$. Formally, for each state holds $\vec{q} \in \mathcal{C}$ and the configuration space of a fixed robot in three-dimensional space creates a six-dimensional manifold.

Since $\mathcal{C}$ contains all possible configurations of the substrate (robot $\mathcal{R}$) in $\mathcal{W}$ it as well contains the states that represent situations in which the substrate will collide with a channel wall. The motion planning theory divides the configuration space into two subspaces, which are called the *obstacle space* and the *free space* denoted as $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ respectively. The obstacle space contains all states that represent cases when the robot have a collision with an obstacle and the free space contains the rest of them.

Formally, let the $\mathcal{O} \subset \mathcal{W}$ be an obstacle; $\mathcal{R} \subset \mathcal{W}$ be a substrate and $\vec{q} \in \mathcal{C}$. If the substrate position in the world space is denoted by $\mathcal{R}(\vec{q})$ then the $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ can be defined as:

$$\mathcal{C}_{obs} = \{\vec{q} \in \mathcal{C} \mid \mathcal{R}(\vec{q}) \cap \mathcal{O} \neq \emptyset\} \quad (1)$$

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs} \quad (2)$$

To solve the motion path planning problem it is essential to find a connected path $\mathcal{P}$ from a starting state $\vec{q}_s$ to a goal state $\vec{q}_g$. Furthermore, the path has to satisfy $\forall \vec{p} \in \mathcal{P} \mid \vec{p} \in \mathcal{C}_{free}$ otherwise the solution would not be valid (see Figure 1).

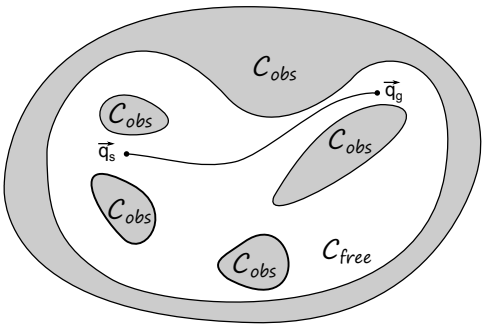


Figure 1: The connected path from $\vec{q}_s$ to $\vec{q}_g$.

Generally, there are two possible approaches to solve this problem. The *combinatorial approach* solves the problem without any approximations. The basic idea is to evaluate the whole free and obstacle space and then find a connected path $\mathcal{P}$ from $\vec{q}_s$ to $\vec{q}_g$ that lies completely in $\mathcal{C}_{free}$. Unfortunately this method has a high computational complexity and therefore is not sufficient for solving practical problems.

Most of the current algorithms are therefore using the *sampling-based approaches*. The configuration space is sampled (usually in a deterministic way) into a finite set of samples. The main concept of this approach is to find only a limited set of points on the path $\mathcal{P}$ instead of the explicit construction of the whole configuration space. The connected path is computed by an interpolation between the nearby samples. On one hand, the complexity of this method is lower but on the other hand the accuracy is lower as well. Both factors usually depend on the number of

samples – the lower number produces more errors while the higher number increases the computational time.

4 Algorithm

In this section we present a new algorithm based on the motion path planning that alleviates the need for a bounding volume approximation of a substrate.We will first describe the algorithm background, then provide an overview of the algorithm after which the necessary conditions for boundary configurations and three running modes of the algorithm will be described. The primary goal of the algorithm is to detect whether a given substrate can pass through a circle. The input of the algorithm is a set of spheres defining the substrate geometry and the radius of the circle through which the substrate should pass.

4.1 Algorithm Background

As shown in Figure 2, the radius of a protein channel varies over its length. Our measurements indicate that some parts of the channel will be wide enough for the substrate to pass through in an arbitrary configuration $\mathcal{R}(\vec{q})$. However, in the real scenario this does not hold for the whole length of the channel. The narrower parts of channel (*gaps*) create channel bottlenecks where the substrate will collide with the channel "wall" (the atoms of the protein) in some configurations or will not pass through them at all. This basically means that if the molecule can pass through all these gaps it can as well pass through the whole channel.

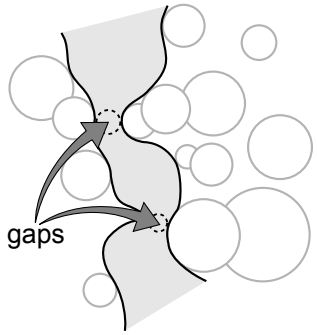


Figure 2: Gaps in the protein channel (adapted from [3]).

Hence we can approximate the problem of substrate passing through the whole channel to the problem of substrate passing through a set of gaps. Moreover, the protein channel can be approximated as a circle-profile shaped tunnel [3] and we can simplify the previous problem to the problem of the substrate passing through a set of circles.

Our algorithm solves the passing problem for each circle individually. The basic idea is to find a sequence of rotations and shifts that will result in the successful passing of the given substrate through the circle, from one side to the other. In this section, we will show that it can be done by using the motion path planning. Note, that if there are two or more circles close enough, we can generalize the problem and for the second circle start in the middle of the passage provided that we have appropriately modified the collision detection and marked all already passed atoms.

The problem of passing substrate through the circle, however, can be inverted to a problem of pulling the circular ring over the static substrate, from one side to the other. Both concepts are almost identical except that the transformations will be inverted. The second view is easier for explanation of the algorithm and therefore will be used in this paper.

Note that $\mathcal{R}$ from now on will denote the new circle robot and the obstacles set $\mathcal{O}$ refers to the spheres defining the substrate geometry. We can also use the fact that the circle robot $\mathcal{R}$ is invariant under the rotation around one of the axis. Hence, the state defining the circle position $\mathcal{R}(\vec{q})$ in the space can be actually reduced to five-dimensional vector $\vec{q}(x_t, y_t, z_t, u_t, v_t)$. Obviously, the reduction of the state-space dimensionality leads to the significant speedups.

4.2 Molecular Pass Planning

We will start with an overview of the algorithm and describe it in detail later. The main idea of this algorithm is to incrementally compute the continuous path of the circle over the substrate from already known configurations.

Pseudocode 1 Molecular pass planning algorithm.

Input: a set of spheres (connected by red and green lines), the circle radius
Output: a connected path of the circle over the substrate

```
1:  find the starting configuration R(q_s)
2:  S ← R(q_s) {S is a stack}
3:  loop
4:    if S is empty then
5:      exit ⟶ fail: the circle is to small
6:    else
7:      R ← S.top()
8:      R.markAsUsed()
9:      R.previous.successor ← R
10:     if R satisfy conditions for R(q_g) then
11:       end loop ⟶ success: there is a path P
12:     else
13:       find the next possible configurations N from
          R according to the mode and put each n ∈ N
          onto the stack S
14:     end if
15:   end if
16: end loop
17: create a connected path P using .successor attributes
```

We have observed that: If there is a connected collision-free path $\mathcal{P}$ of the circle $\mathcal{R}$ from one side of the substrate $\mathcal{O}$ to the other we can cut the substrate at any time by the hyperplane containing $\mathcal{R}$. The cut will then produce a set

of additional circles, which represent the crosscuts of the atoms lying in the cut plane. These circles must not collide with the circle $\mathcal{R}$ otherwise the path $\mathcal{P}$ would not be collision-free (see Figure 3).

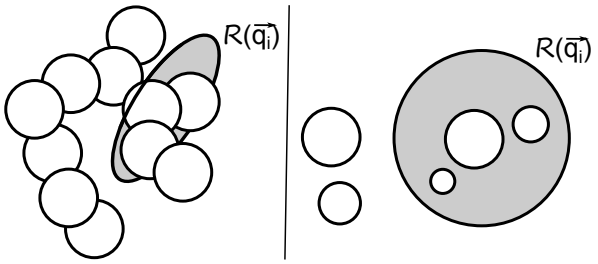


Figure 3: The crosscuts of atoms lying in the hyperplane containing $\mathcal{R}$.

We utilize the fact that all atoms of the substrate have to pass through the circle. In other words each atom has to cross the hyperplane containing the circle $\mathcal{R}$. Therefore we can sample the path $\mathcal{P}$ at this moments which with the previous observation allows us to use the sampling-based version of the motion path planning.

To create a connected path $\mathcal{P}$ of the circle $\mathcal{R}$ over the substrate we need to find a sufficient number of samples and then interpolate between them. The interpolation between two configurations has to be collision-free as well. To achieve that we need to test whether the computed path of the circle will not collide with any atom of the substrate.

This can be done either by the continuous collision detection or by using the sampling-based approach. The continuous collision detection is very computationally demanding in this case because we have to compute the intersections of a collapsed torus with a set of spheres. Hence, we decided to use the sampling approach based on collision detection between the circle and a set of spheres.

Additionally, for a valid detection whether the substrate can pass through the protein channel we need to test it for collisions with the atoms of the protein as well.

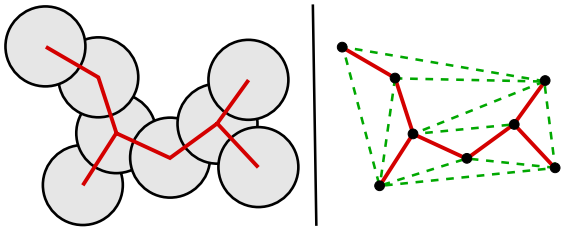


Figure 4: Left: The substrate molecule as a set of spheres connected by red (solid) lines. Right: The ball-and-stick model with the Delaunay triangulation represented by green (dashed) lines.

The approximation of the substrate as a set of spheres is simple enough for computing all collisions during the interpolation process. However, this model is not suitable for the other part of the algorithm. To be able to compute all samples efficiently we need to consider the structure of the substrate, namely bonds between atoms. The appropriate data structure is based on the ball-and-stick model with additional information stored in *green lines* (see Figure 4). The extra lines represent the edges of the Delaunay triangulation computed on the atoms of the substrate and are used for the samples computing.

Our algorithm employs the sample-based approach. As we are only interested in the knowledge whether the substrate can pass through the circle, we can use an incremental sampling. In other words, we do not compute all samples in advance but we compute the first configuration $\mathcal{R}(\vec{q}_s)$ and then compute a next possible configuration $\mathcal{R}(\vec{q}_{s+1})$. If there are more then one successor configurations we will pick one randomly and push the rest onto the stack for further processing. The whole process is then repeated until we reach the goal configuration $\mathcal{R}(\vec{q}_g)$ that represents the case when the substrate had passed through the circle, or we end if there is no valid successor of the previous configuration.

The algorithm is similar to a depth-first search for traversing a tree structure. However, the algorithm does not need the tree structure itself. The only two needed structures are the stack S and the hash table for remembering already visited configurations, to avoid cycling.

The successor configurations are computed incrementally from the known configurations. The successor configuration $\mathcal{R}(\vec{q}_{i+1})$ can be created from the previous configuration $\mathcal{R}(\vec{q}_i)$ by replacing one of it's atoms. The configuration can be represented either by one, two or three atoms, see section 4.4. The conditions for the replaced atom in $\mathcal{R}(\vec{q}_i)$ and for the new one in $\mathcal{R}(\vec{q}_{i+1})$ are as follows:

For the atom that will be replaced, there is no limitation at all. In other words we can replace any atom in the set. From practical point of view, however, the choice can influence whether the incrementally built path $\mathcal{P}$ will lead to the successful passing of the circle over the substrate. Therefore, during the search, we need to compute the whole set of successor configurations for every atom. Then we randomly pick one and store the remaining configurations for eventual processing in the future.

The new atom has to be connected by either red or green line with the atom which will be replaced. This condition will assure that we will not try to make too long jumps. The short steps are necessary because between two distant configurations we cannot easily interpolate to create a connected path.

4.3 Boundary configurations

The motion path planning solves the problem of finding a connected path from $\vec{q}_s$ to $\vec{q}_g$. The starting configuration $\mathcal{R}(\vec{q}_s)$ has to satisfy two conditions. Firstly, the plane containing the circle in starting configuration $\mathcal{R}(\vec{q}_s)$ should cut only one atom. This condition is not compulsory, its purpose is to simplify the problem of correct orientation of

the first configuration in the space. Secondly, all remaining atoms of the substrate have to lie on the same side from the plane defined by the circle.

The problem, that can occur when violating the second condition, is illustrated in Figure 5. If we select the red atom $\mathcal{A}$ as a starting point, the configuration would assume that some atoms had already passed through the gap. To avoid this, we reduce the set of possible starting atoms to atoms lying on the convex hull of the substrate (see Figure 5 right).

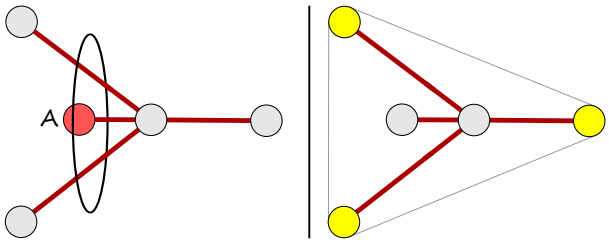


Figure 5: Left: The unappropriate atom choice. Right: The convex hull of the substrate.

However this will not solve the problem completely. We also have to be careful about the size of the circle. Because even though that we choose the starting atom at the convex hull a problem can occurs. There can be another atom $\mathcal{B}$ such that the intersection of its connection line and the hyperplane containing the circle $\mathcal{R}$ will lie inside the circle $\mathcal{R}$ (see Figure 6). This is the case where the second condition will be violated.

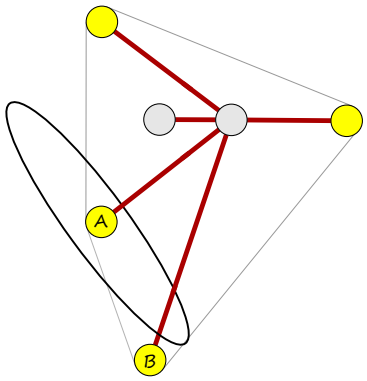


Figure 6: The problem with the convex hull.

In this case, we have to explicitly check whether all atoms of the substrate are lying on one side of the circle defined by the configuration $\mathcal{R}(\vec{q}_s)$.

Similar to the starting configuration there can be more than one end configuration. The conditions and rules that affect the end configuration are almost the same. The difference is that all atoms have to be on the other side of the circle than at the beginning.

4.4 Modes

The algorithm has to deal with all types of molecules. Therefore we propose to use it in three modes, *1D, 2D* and *3D*, each named by a number of atoms they are using for a configuration representation. The 4D or higher modes are not necessary because in real scenarios there are only rare occasions when four or more nearby atoms are lying in one plane and if they do, the situation can be handled by two or more subsequent 3D configurations. Algorithm selects the current mode according to the complexity of the substrate in the specific area. It is possible to switch between modes by adding/removing one atom to/from the current configuration. In one algorithmic step, it is possible to switch between 1D and 2D, or 2D and 3D modes only.

4.4.1 1D Mode

The first mode is responsible for handling polymer-like molecules or parts of complex molecules where atom bonds can be represented as a polygonal chain (Figure 7).

There is always at least one starting configuration represented by a single atom (see section 4.3). Let the current configuration $\mathcal{R}^{1D}(\vec{q}_i)$ be such configuration. At this point the algorithm has two basic possibilities how to compute the successor configuration. It can be another configuration represented by one atom $\mathcal{R}^{1D}(\vec{q}_{i+1})$ or the algorithm can switch from the 1D mode to the 2D mode. The 2D configuration $\mathcal{R}^{2D}(\vec{q}_{i+1})$ is then created by adding an additional atom into the current configuration $\mathcal{R}^{1D}(\vec{q}_i)$.

Pseudocode 2 The 1D mode traversing.

Input: $\mathcal{C}_i$ – current configuration $\mathcal{R}^{1D}(\vec{q}_i)$ from stack S
Output: a successor configuration $\mathcal{C}_{i+1}$

```
1:  A ← C_i.getAtom()
2:  N ← A.getAllNeighbours()
3:  for all B in N do
4:      if B is a complex atom then
5:          C_{i+1} ← the new 2D configuration represented by
            the atoms A and B
6:      else
7:          C_{i+1} ← the new 1D configuration placed in B
8:      end if
9:      if C_{i+1} was not used and a collision-free path from
        C_i to C_{i+1} exists then
10:         C_{i+1}.previous ← C_i
11:         put C_{i+1} onto the stack S for further processing
12:     end if
13: end for
```

In this mode the configuration $\mathcal{R}(\vec{q}_i)$ is represented by one atom $\mathcal{A}$. It is clear that one atom (or rather one point) cannot define the orientation of the circle $\mathcal{R}$ in the three-dimensional space. Hence, we need to define normal vector $\vec{n}$ of the circle $\mathcal{R}$. We utilize the fact that atoms of the substrate's molecule are connected. This mean that every

atom $\mathcal{A}$ has at least one neighbor $\mathcal{B}$ and we denote this connection by the red line. We call the vector $\overrightarrow{\mathcal{AB}}$ a *segment*. The normal vector $\vec{n}$ of the circle $\mathcal{R}$ then corresponds to the segment originating from the atom by which the configuration is represented – in this case atom $\mathcal{A}$ (see Figure 7).

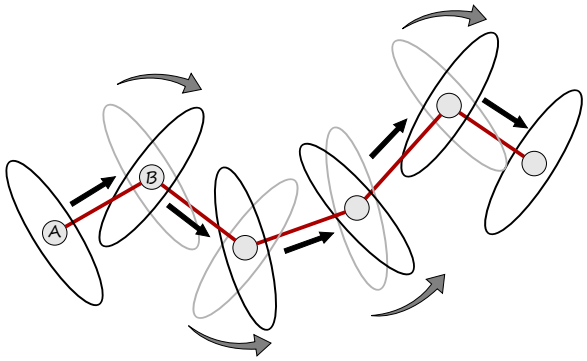


Figure 7: The 1D mode traversing.

As shown in Figure 7, the interpolation between two 1D configurations is a simple translation along the segment and a rotation at the end. The rotation is necessary to orient the circle properly before traversing to the next segment of the polynomial curve defining the substrate.

Whether the successor configuration will be 1D or 2D is determined by the complexity of the molecular structure in the specific part of the substrate. More precisely, if the current atom $\mathcal{A}$ has more then two neighbors or there is at least one green line connecting $\mathcal{A}$ with another atom, then our algorithm will switch to the 2D mode. In this case we call the atom $\mathcal{A}$ a *complex atom* (see Figure 8). Otherwise the algorithm will stay in the current mode and will compute another 1D configuration. The switch from 1D to 2D is done simply by including one of the neighbor atoms connected to $\mathcal{A}$ with red (solid) or green line (dashed).

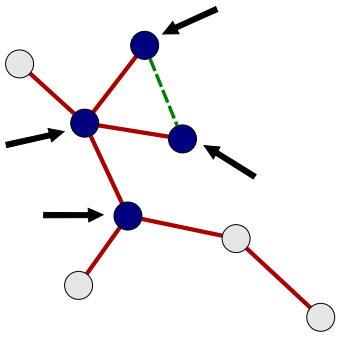


Figure 8: Complex atoms in substrate model.

4.4.2 2D Mode

While in the 1D mode the circle moves from one configuration to another by translation, in the 2D mode the path is built by rotations. The example of such rotational move is shown in Figure 9. The algorithm proceeds from the state $\vec{q}_i$ that represents the position $\mathcal{R}^{2D}(\vec{q}_i)$ of the circle in space and searches for a configuration $\mathcal{R}^{2D}(\vec{q}_{i+1})$. The successor configuration in the 2D mode can be computed by replacing one of the atoms $\mathcal{A}$ or $\mathcal{B}$ by a new atom $\mathcal{C}$. If the smallest circle containing the triangle $\triangle\mathcal{ABC}$ is smaller than $\mathcal{R}$, the algorithm will switch to the 3D mode.

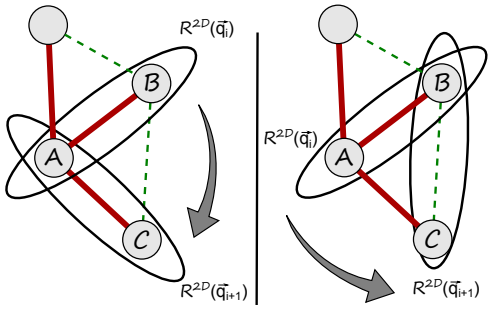


Figure 9: The 2D mode traversing.

Obviously, in 2D mode there may be more then one successor configuration $\mathcal{R}^{2D}(\vec{q}_{i+1})$ If such situation occurs, the algorithm randomly selects one of the possible configurations and others are stored for later processing.

Pseudocode 3 The 2D mode traversing.

Input: $\mathcal{C}_i$ – current configuration $\mathcal{R}^{2D}(\vec{q}_i)$ from stack S
Output: a set of successor configurations $\mathcal{C}_{i+1}$

```
Atoms ← C_i.getAllAtoms()
for all A in Atoms do
    {note that the second atom in C_i is denoted B}
    N ← A.getAllNeighbours()
    for all C in N do
        if the smallest circle containing △ABC is smaller
        then R then
            C_{i+1} ← the new 3D configuration represented
            by the atoms A, B and C
        else
            C_{i+1} ← the new 2D configuration represented
            by the atoms B and C
        end if
        if C_{i+1} was not used and a collision-free path
        from C_i to C_{i+1} exists then
            C_{i+1}.previous ← C_i
            put C_{i+1} onto the stack S
        end if
    end for
end for
```

The 2D configuration $\mathcal{R}^{2D}(\vec{q}_i)$ is represented by two atoms $\mathcal{A}$ and $\mathcal{B}$. The center is situated in the middle of the line connecting the atoms. To represent a circle orientation in the three-dimensional space, we need to compute the normal vector $\vec{n}$.

Let the current configuration $\mathcal{R}^{2D}(\vec{q}_i)$ be defined by two atoms $\mathcal{A}$ and $\mathcal{C}$ and the previous configuration $\mathcal{R}^{2D}(\vec{q}_{i-1})$

be defined by atoms $\mathcal{A}$ and $\mathcal{B}$. The normal vector $\vec{n}$ of the circle $\mathcal{R}^{2D}(\vec{q}_i)$ can be then computed as a cross product $\overrightarrow{\mathcal{AC}} \times \vec{R}$, where $\vec{R}$ is a vector of the rotation from $\mathcal{R}^{2D}(\vec{q}_{i-1})$ to $\mathcal{R}^{2D}(\vec{q}_i)$ and can be computed as $\overrightarrow{\mathcal{AB}} \times \overrightarrow{\mathcal{AC}}$. That gives us the equation for the normal vector:

$$\vec{n} = \overrightarrow{\mathcal{AC}} \times (\overrightarrow{\mathcal{AB}} \times \overrightarrow{\mathcal{AC}}) \tag{3}$$

Note that the orientation of this normal depends on the order of the vectors in the equation. However, if we assume that the rotation from $\mathcal{R}^{2D}(\vec{q}_{i-1})$ to $\mathcal{R}^{2D}(\vec{q}_i)$ is done with the smallest possible angle then the correct orientation of the $\vec{n}$ can than be ensured by a simple rule. We denote the half-spaces induced by the plane containing $\mathcal{R}$ according to Figure 10 and let a point p be the center of the line connecting the centers of both configurations. In order to ensure the correct orientation of the normal $\vec{n}$ the point p has to lie in a differently signed half space.

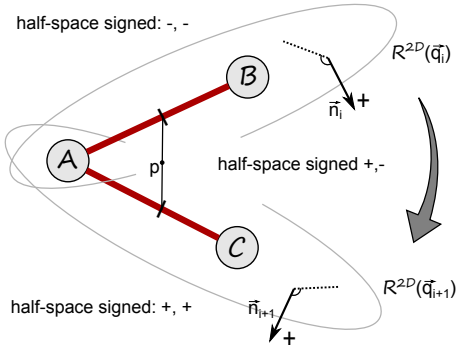


Figure 10: Space divided by two configurations.

Similarly to the 1D mode, it is necessary to apply a small position correction between each step. While in the 1D mode we needed a rotation to put the circle into the right orientation before we could traverse the next segment, here it is a shift which solves the problem of different emplacement of the circle centers. If we simply rotate from $\mathcal{R}^{2D}(\vec{q}_i)$ to $\mathcal{R}^{2D}(\vec{q}_{i+1})$ we would get a different circle center than we have originally computed by replacing atom $\mathcal{B}$ by atom $\mathcal{C}$ (see Figure 11). Hence, before we rotate or after rotation we have to shift the appropriate configuration into the right position.

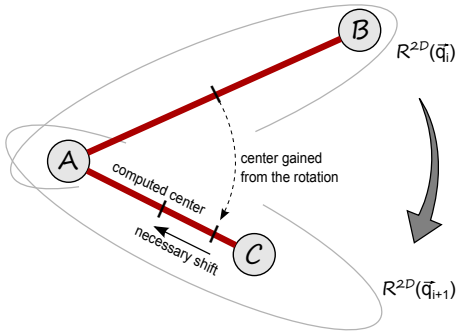


Figure 11: The position correction between each step.

4.4.3 3D Mode

The difference of the 3D mode compared to the previous is that circle in this mode is represented by three atoms $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. This mode provides more degrees of freedom for the rotations between configurations then the previous ones. Additionaly, we do not need to compute the normal vector $\vec{n}$. The center of the circle is defined as a center of the smallest circle containing the triangle $\triangle ABC$. The problem caused by center difference during the passing from $\mathcal{R}^{3D}(\vec{q}_i)$ to $\mathcal{R}^{3D}(\vec{q}_{i+1})$ is the same as in the 2D mode and it may be also solved by applying translation before the rotation.

Pseudocode 4 The 3D mode traversing.

Input: $\mathcal{C}_i$ – current configuration $\mathcal{R}^{3D}(\vec{q}_i)$ from stack S
Output: a set of successor configurations $\mathcal{C}_{i+1}$

```
Atoms ← C_i.getAllAtoms()
for all A in Atoms do
    {the remaining atoms in C_i are denoted B and C}
    N ← A.getAllNeighbours()
    for all D in N do
        if the smallest circle containing △BCD is smaller
        then radius of R then
            C_{i+1} ← the new 3D configuration represented
            by the atoms B, C and D
        end if
        if C_{i+1} was not used and a collision-free path
        from C_i to C_{i+1} exists then
            C_{i+1}.previous ← C_i
            put C_{i+1} onto the stack S
        end if
    end for
end for
```

5 Results

The algorithm was tested nearly on the 250 real ligands downloaded from the protein database[1]. The number of atoms in ligands used for the testing varied from 5 to 168. The starting point for each ligand was chosen by a user to satisfy conditions described in the section 4.3. Using binomial search we determined the smallest circle for which the algorithm would still return positive answer on question whether the ligand would pass through it or not. The radius of the smallest circle was then compared with the radius of the bounding sphere. The final results are shown in Figure 12. According to these results we are able to find a path through a 40% (in average) narrower channel in comparison with the bounding sphere approach.

The number of steps needed to compute the path of a ligand from one side of the circle to the other one is shown in the next graph (Figure 13). Our experiments show that

[1] http://www.rcsb.org/pdb/home/home.do

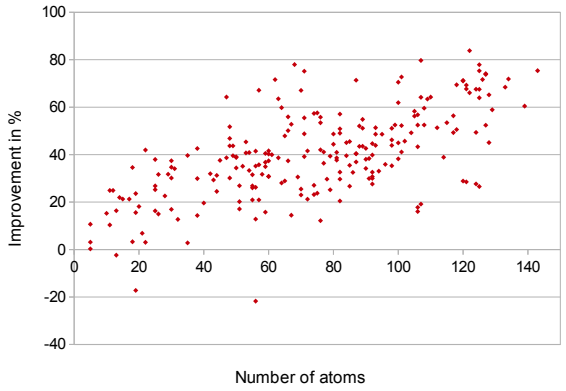


Figure 12: The improvement of the channel radius in comparison to the bounding sphere method.

the average time complexity for the ligands used for testing tends to be between $O(n\ ln\ n)$ and $O(n\ ln^2\ n)$ where n denotes the number of atoms in ligand.

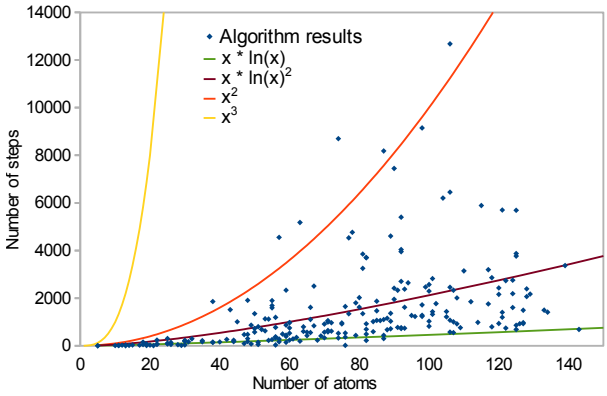


Figure 13: The number of steps needed to compute the path of the ligand through the smallest possible gap represented by a circle.

As it can be seen in Figure 12, our algorithm has some limitations. In some cases the method provides the worse solution than the bounding sphere method (see the points with negative values). These cases occur due to the fact that in our implementation the circle is using only a simple hard-coded sequence of a translation followed by a rotation. The collision detection on the molecule structure can then, in particular cases, prevent the circle from switching from one configuration to another even though the different sequence of movements would allow it.

We believe that this problem can be removed entirely by implementing a more complex system of movements. The proposed algorithm, however, would still not be optimal in the sense of minimizing the circle radius. The radius of the minimal circle found by our algorithm is limited by the distance of the connected atoms (see Figure 14).

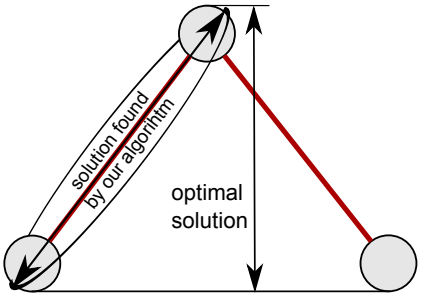


Figure 14: The limitations of the presented algorithm.

6 Conclusion

We have described a simple and fast algorithm based on the motion path planning that computes a path of a set of spheres through a circular-profile shaped gap. Three modes of the algorithm was purposed. Each of them solving part of the given problem according to the complexity of the substrate in the specific area. It was shown that the purposed algorithm has the average time complexity somewhere between $O(n\ ln\ n)$ and $O(n\ ln^2\ n)$ where n denotes the number of atoms in ligand and can find a path through a 40% (in average) narrower channel in comparison with the bounding sphere approach.

Acknowledgements

I would like to thank to my colleague Mgr. Petr Tobola, Ph.D for his original idea which is the base of my diploma thesis and this paper.

References

[1] Maciej Haranczyk and James A. Sethian. Navigating molecular worms inside chemical labyrinths. *Proceedings of National Academy of Science (PNAS)*, pages 21472–21477, 2009.

[2] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[3] Peter Medek, Petr Beneš, and Jiří Sochor. Computation of tunnels in protein molecules using delaunay triangulation. *Journal of WSCG*, pages 107–114.

[4] Amit P. Singh, Jean-Claude Latombe, and Douglas L. Brutlag. A motion planning approach to flexible ligand binding. *ISMB-99 Proceedings*, pages 252–261.

[5] Guang Song and Nancy M. Amato. A motion-planning approach to folding: from paper craft to protein folding. *Robotics and Automation, IEEE Transactions on*, 20(1):60 – 71, feb. 2004.

Finding Cavities in a Molecule

Lukáš Jirkovský
Supervised by: Martin Maňák and Ivana Kolingerová

Department of Computer Science and Engineering
University of West Bohemia
Pilsen / Czech Republic

Abstract

In recent years, biochemistry is gaining more and more attention. The research involves analysis of large molecules, such as proteins. One of many properties we can study are molecular cavities, where cavity is understood as a free space inside a molecule. As we are usually interested only in a certain subset of cavities, the common approach is to use a spherical probe of a given radius to find the cavities. The probe can be imagined as a sphere which we try to slip through the molecule.

In this paper we discuss an algorithm to find inner cavities in a molecule for a given size of the probe. The algorithm has a preprocessing stage where an additively weighted Voronoi diagram of a molecule is computed. This diagram is then used to accomplish the task of finding cavities for varying probe sizes.

The algorithm presented proved to be very fast for a probe with a variable size. The implementation shows it is able to operate in real-time even on large structures, such as the Thermus Thermophilus 70S ribosome (PDB ID 3OH5, approximately 87 000 atoms).

Keywords: molecular cavity, molecule analysis, additively weighted Voronoi diagram

1 Introduction

A protein structure can be very complicated. This includes depressions on the molecular surface (often called pockets) and empty space inside the molecule. This empty space can form tunnels and cavities which are not connected to the surface (inner cavities). It can be represented as a union of spheres (Figure 1). When searching cavities, we usually introduce a spherical probe which we try to slip through the molecule. This is useful, because it allows us to specify the minimal radius of the cavity (the radius is usually measured in angstroms [Å], where $1\text{Å} = 10^{-10}$ m).

The protein structure affects the behavior of protein interactions. These interactions are a part of biological processes. This has led to the study of protein structures and using the knowledge of these structures to design new drugs. A lot of research has been done on so-called active sites, which are the places where the proteins can mutually interact. These sites are generally found on the surface of a protein. However, as protein molecules are not perfectly stationary, some of the inner cavities may become accessible from the outside at some point of time and become a part of an active site. Therefore, it is desirable to identify these cavities.

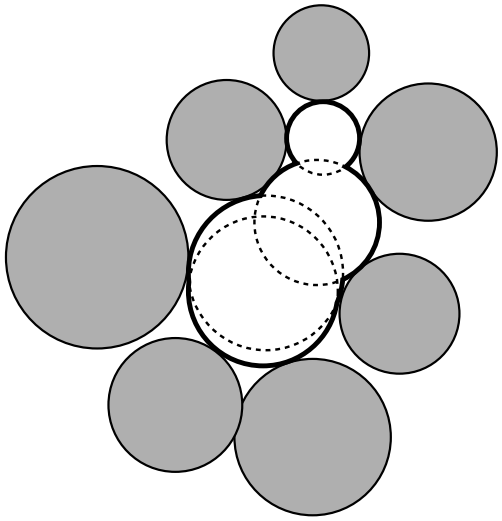


Figure 1: Cavity formed by gray atoms.

The inner cavities can also hold buried residui of other molecules, such as water [21, 23]. This is important as it can influence the stability of protein structure [23].

The existing cavity searching algorithms are proficient in searching the cavities, however, they are slow when it comes to a variable probe size, because they need to rerun all computations when the probe size changes. Our method improves the run time when different radii of probe are used by moving some of the computational complexity into the preprocessing stage. The preprocessing allows the algorithm to operate very quickly on large molecules (about 100 000 atoms).

Our algorithm has been implemented as a Java library, because in the future we would like to use it as a plugin in CAVER [2], a software tool for protein analysis and visualization.

In Section 2 the current methods used for finding inner cavities will be presented. In Section 3 the necessary the-

oretical background of the ordinary Voronoi diagram and the additively weighted Voronoi diagram with its dual representation will be given. In Section 4 the algorithm using the additively weighted Voronoi diagram for finding inner cavities will be described. In Section 5 we will present results of our implementation. Finally, Section 6 summarizes our findings.

2 State of the Art

The algorithm to find cavities in a molecule using alpha-shape [6] has been presented by Liang et al [15, 16]. First, the radii of all atoms are increased by the radius of probe to obtain the "solvent accessible" model [14, 15] – the model which is accessible by the given probe. Then a regular triangulation of atom centers is computed and the cavities are searched within the weighted alpha shape.

The weighted alpha shape is a subset of the regular triangulation. This subset is determined by the parameter α, which can be imagined as the radius of a probe is rolling over the molecular surface. This probe removes edges and thus collapses some of the tetrahedra.

The weighted alpha shape used in this algorithm is constructed by removing edges which are dual to Voronoi vertices (see Section 3) outside of the enlarged molecule atoms. This corresponds to using a probe of a zero radius. It has been shown [5] that voids in the alpha shape correspond to cavities in a molecule. The cavities are identified by searching the tetrahedra that were removed when the alpha-shape was constructed. Only the tetrahedra which were completely enclosed in the alpha shape are considered. The downside of this algorithm is that the alpha-shape has to be rebuilt whenever the probe size is changed.

Several algorithms using a regular grid exist [7, 8, 22]. The first step of these algorithms is construction of a regular grid, where each grid cell stores information whether it lies inside or outside of the molecular surface. The grid is then processed to identify cavities. The processing depends on the algorithm.

The algorithm described in [7] identifies cavities by checking the neighboring cells of an empty cell in the direction of each axis. The algorithm described in [8] identifies cavities by scanning the grid in the direction of each axis and diagonals. Another approach was presented by Tripathi and Kellogg [22] as the VICE algorithm (Vectorial Identification of Cavity Extents). This algorithm constructs a set of vectors from every empty grid cell. The visibility of the molecular surface for each vector is determined. The visibility describes how much the grid cell is enclosed within the molecule. The inner cavities are formed by points fully enclosed within the molecule.

Our algorithm uses the additively weighted Voronoi diagram in the preprocessing stage. This diagram has many other uses apart from searching cavities, such as finding pockets [9] or determining how spherical the molecule is [13]. The algorithm for the construction of the diagram has been described in [11, 12, 18]. To improve the speed of the edge-tracing algorithms for the diagram construction, spatial filtering is often used [17, 24, 3].

3 Geometric Background

To fully understand the idea of the algorithm, some of the properties of Voronoi diagrams need to be mentioned first. We will begin with the description of an ordinary Voronoi diagram. Then a basics of the additively weighted diagram and its dual representation called quasi-triangulation will be given.

3.1 Voronoi Diagram

A Voronoi diagram [19] is a decomposition of the space determined by a set of points. These points are often called *generators*. We can describe the Voronoi diagram as a tessellation of space, such that for every generator we define a *Voronoi region*, consisting of all points in space which have the smallest Euclidean distance to its generator. That means for each generator p_i there exists a Voronoi region R_i, such that:

$$R_i = \{x : \|p_i - x\| \leq \|p_j - x\|, \forall j \neq i\}$$

where $i, j \in \{1, 2, \ldots n\}$ and n is the number of generators. For an example of a two-dimensional Voronoi diagram see Figure 2.

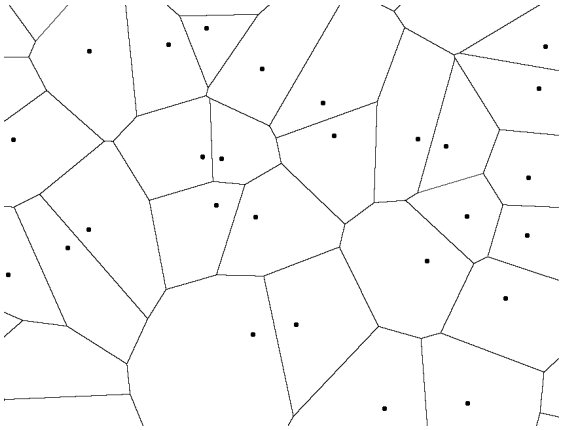


Figure 2: Two-dimensional Voronoi diagram.

Region boundaries are called *Voronoi edges*. It can be seen that in the two-dimensional Voronoi diagram each edge is shared among neighboring regions. In the three-dimensional Voronoi diagram the Voronoi edge is shared among three regions. Each point of an edge is equidistant to the edge generators, because the points lying on the edge have to meet the definition of Voronoi region for all regions containing the edge.

The point where multiple edges meet is called a *Voronoi vertex*. A Voronoi vertex can be also defined as the endpoint of the Voronoi edge. In two dimensions we can

also say that the Voronoi vertex is the point shared among three Voronoi regions. Similarly, in the three-dimensional Voronoi diagram the Voronoi vertex is a point shared by four regions.

The Voronoi diagram is usually seen as a graph of the Voronoi edges and the Voronoi vertices. The diagram is often stored in its dual representation, called the Delaunay triangulation. Edges in the triangulation are created by connecting the closest pair of generators, i.e., generators that share an edge. As a result we obtain a triangle in the Delaunay triangulation for each Voronoi vertex in two dimensions and a tetrahedron in three dimensions.

3.2 Additively Weighted Voronoi Diagram

It can be seen that all generators affect the resulting Voronoi diagram equally when the Euclidean metric is used. However, this is not always desired, consequently many variations exist. One of these variations is the additively weighted Voronoi diagram, also known as the Apollonius diagram or the Euclidean Voronoi diagram of spheres.

Each generator in the additively weighted Voronoi diagram has a weight. Unlike the ordinary Voronoi diagram, where generators are imagined as points, generators in the additively weighted Voronoi diagram are imagined as circles in 2D or spheres in 3D with the radius equal to their weight.

An *additively weighted distance*, which is defined as the Euclidean distance minus the weight of the generator [19, 11], is used instead of the Euclidean distance. Let $i, j \in \{1,2,\ldots n\}$, where n is the number of generators. We define the Voronoi region R_i for a generator p_i with a weight w_i as:

$$R_i = \{x : \|p_i - x\| - w_i \leq \|p_j - x\| - w_j, \forall j \neq i\}$$

Due to the used distance, the Voronoi region can be interpreted as a set of points closest to the sphere p_i with the radius w_i. This has a few important implications. The edges are no longer necessarily line segments or half-lines. For an example of the additively weighted Voronoi diagram in two dimensions, see Figure 3.

Unfortunately, some anomalies can occur in the additively weighted Voronoi diagram. It is possible that two generators define more than one edge if there is a generator with a small weight among the generators with a big weight assigned (e.g., a small sphere among three large spheres). This small generator can split the edge into two parts which are connected by the edges generated by the small generator and the big generators. The generator with a small weight can also generate an elliptic edge (Figure 4). For more details about these anomalies see [12].

Similarly to the ordinary Voronoi diagram, a dual representation exists. This representation is called a quasi-triangulation [12]. This dual representation is not a valid triangulation (hence quasi-), because the anomalies break

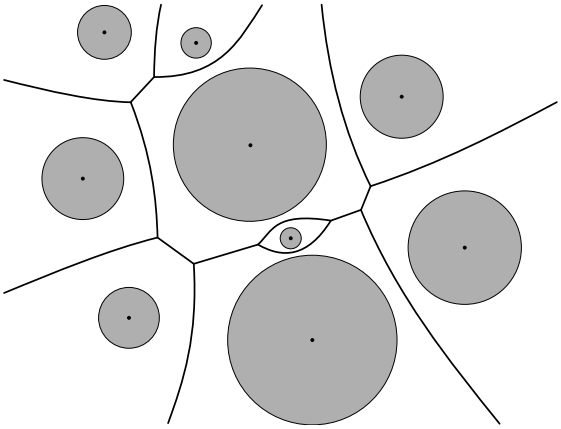


Figure 3: 2D additively weighted Voronoi diagram.

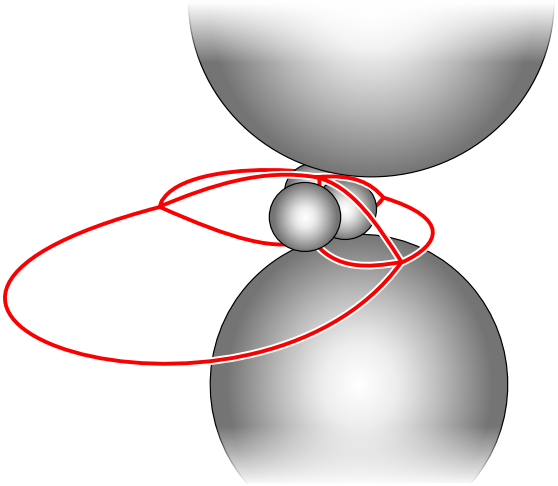


Figure 4: Eliptic edges in the 3D additively weighted Voronoi diagram.

the validity of the triangulation. For example, if an elliptic edge exists in the three-dimensional additively weighted Voronoi diagram, it may not be possible to construct a tetrahedron, because no Voronoi vertex lies on the edge. A triangle is stored in this case. Another example of an anomaly is that a split edge in three dimensions is represented by multiple tetrahedra with two or more common triangles.

4 Proposed method

Our method finds inner cavities in a molecule. Its input is a set of molecule atoms, which are represented as spheres. The output is a set of subgraphs of the graph representing the additively weighted Voronoi diagram. Each subgraph forms a connected cavity, as the probe can be slipped along the subgraph.

We can leverage the additively weighted Voronoi diagram for computing a diagram of molecules. The additively weighted Voronoi diagram, where the generators are

atoms of a molecule with the weight given by their radius, has an important feature, which can be used to quickly find cavities using a probe of a given size. It is the fact that an edge in such a diagram can be considered as an optimal path among the generators.

A very simple proof is that if we moved away from one generator, we would always get nearer to at least one other generator. We can use this knowledge to slip a spherical probe through the molecule. If the probe was not moved along the edge, it would be possible that the probe "hits" one of the generators, while there was a space between the probe and another generator. Moving the probe along the Voronoi edge ensures that the distance to the nearest generators, which are likely to collide with the probe, is equal.

The algorithm is built on the fact that the probe can pass among the generators if and only if it can pass on the edge through the narrowest place. As the algorithm does not need the position where this narrowest place is, only the radius of the probe which can pass among the generators is stored. We will call the radius of the probe on the narrowest place a *bottleneck*. We use the bottleneck to determine whether the probe can slip along an edge. Figure 5 illustrates the idea in two dimensions. This idea also applies in three dimensions.

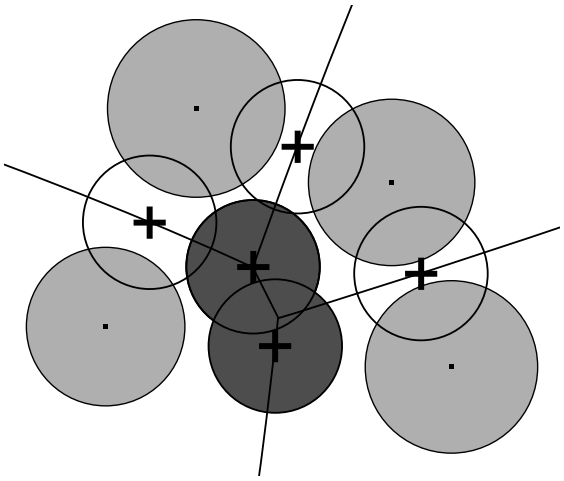


Figure 5: We try to slip a probe among the generators (light-gray circles). The position of the narrowest place is marked by a cross. If the probe can pass through, it is illustrated as a dark circle. If the probe cannot pass through the edge, it is shown as an empty circle.

Our algorithm consists of the following steps:

1. Creation of the additively weighted Voronoi diagram of molecule atoms.
2. Computation of the bottlenecks.
3. Sorting the Voronoi vertices using the distance from the vertex to its generators.
4. Traversal of a diagram using a graph traversal algorithm.

Steps 1–3 are done in a preprocessing step.

4.1 Preprocessing

The algorithm starts with the preprocessing stage. In this stage, the additively weighted Voronoi diagram of atoms is computed. Next, edge bottlenecks are computed, as described later. We also add a boolean flag "is outer" which we will set on for all Voronoi vertices which have at least one edge extending to the infinity. This flag will help us to discover an outer cavity. Finally, Voronoi vertices are sorted by the distance from a vertex to the surface of its generators.

The bottleneck can occur anywhere on the Voronoi edge. To compute the bottleneck we first compute a point that has the minimal distance from the generators defining this edge. Next, we must check whether this point lies on the edge, for which we want to obtain the bottleneck.

This check is done by defining vectors from the center of the generator with the smallest weight to the edge endpoints and a vector to the point with the minimal distance to the generators. The generator with the smallest weight needs to be used because of the elliptic edges. If the vector to the tested point lies within the angle between vectors to the edge endpoints, we store its distance to the surface of the edge generators as the bottleneck for the edge. Otherwise we use the endpoint for which the bottleneck is smaller. Figure 6 illustrates the possible positions of bottlenecks. For details see [11, 18].

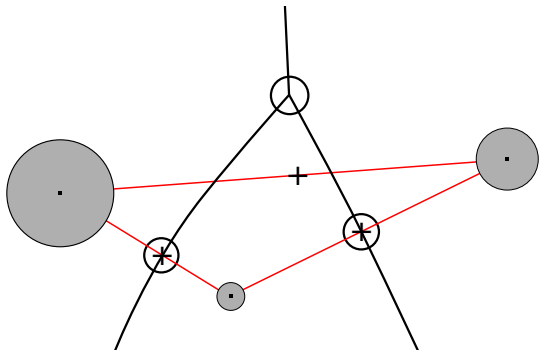


Figure 6: Bottlenecks. Crosses are positions of points with the minimum distance from the edge generators, black circles are actual positions of bottlenecks.

Now, it would be already possible to find the cavities by traversing the graph defined by the Voronoi vertices and edges and checking whether the bottleneck is larger than the probe size. However, this would be very inefficient, because it would require visiting all vertices every time the size of the probe changes.

The efficiency can be greatly improved by sorting the vertices. We store the distance from the vertex to its generators with other vertex information. We call this distance a *maximal bottleneck*. The maximal bottleneck allows us to quickly decide whether the given probe can fit among the generators. Its value is always greater or equal to the maximum of the bottlenecks. This ensures that we cannot

skip any vertex which has large enough bottlenecks during finding cavities.

The last step of preprocessing stage is sorting the vertices using the now obtained maximal bottleneck. The reason to include the distance among the generators and the vertex itself is that we need to take the space among the vertex generators into account. This is important, because the bottlenecks of all edges may be too small, but there is still enough space among the generators of the vertex (see Figure 7).

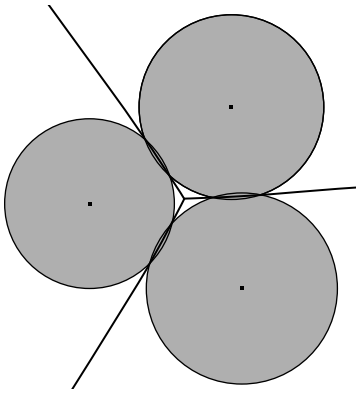


Figure 7: The empty space among generators in 2D. This may occur in three dimensions, too.

4.2 Finding Cavities

The input of this part of the algorithm is the additively weighted Voronoi diagram, represented as a graph of Voronoi vertices and edges and a sorted order of vertices from the preprocessing stage.

The search for cavities is performed as a graph traversal. We start from the vertex with the largest maximal bottleneck. First, we check whether the maximum of bottlenecks of the starting vertex is greater than the probe radius. If this condition is met, we recursively traverse the graph. During the traversal we mark the visited vertices, so we do not visit a vertex multiple times. The traversed part of the Voronoi diagram forms a cavity.

When it is not possible to continue with the traversal, we move to the vertex with the second largest bottleneck and start the traversal from there if possible. We repeat this procedure until a the first vertex is found, for which its maximal bottleneck is large enough for probe to slip through. We can safely stop the algorithm here, as all cavities were found. Since vertices are sorted using their maximal bottleneck, none of the remaining vertices has any bottleneck bigger than the probe radius.

Now, we have found all cavities in the molecule. However, they include the outer void, too. To remove it, we will utilize the flag "is outer" introduced earlier. After the graph has been traversed, we check the "is outer" flag for each visited vertex. If any of the visited vertices has the flag set to true, we disregard this part of the graph, because it is connected to the outer space, hence it cannot be an inner cavity. We can also modify the algorithm to identify first the outer cavity and then search cavities only within the remaining vertices. The advantage of such a modification is that it is possible to use a probe of a different size to remove the outer cavity, which would allow us to use the algorithm to find pockets on the surface of a molecule, too. This is similar to the creation of β-shape [10] prior to the search if the quasi-triangulation was used instead of the additively weighted Voronoi diagram.

5 Experiments and Results

We have developed a Java library implementing the algorithm and a simple visualization tool, the output of which can be seen in Figure 8. The visualization approximates the shape of a cavity by putting spheres into the Voronoi vertices forming the cavity. The implementation uses the awVoronoi library [1] for computing the quasi-triangulation. For that reason the algorithm described had to be converted to the dual representation.

The system used for experiments was a PC with Core i7 920 CPU (four cores at 2.7GHz with Hyper-Threading) and 12 GB RAM running Arch Linux 64bit. In all measurements, we evaluated our algorithm twelve times with the given parameters, removed the shortest and longest measured time and finally computed average of the ten remaining times.

5.1 Algorithm Run Time

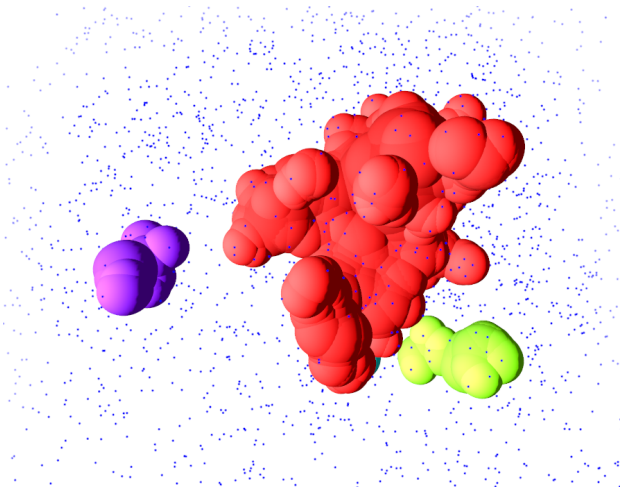
We have evaluated the run time of our algorithm with regard to a variable probe size and a variable molecule size.

Table 1 and Figure 9 shows the measured algorithm run time for the variable size of the probe. We used the Thermus Thermophilus 70S ribosome complexed with chloramphenicol (PDB ID 3OH5, approximately 87 000 atoms) in this experiment.

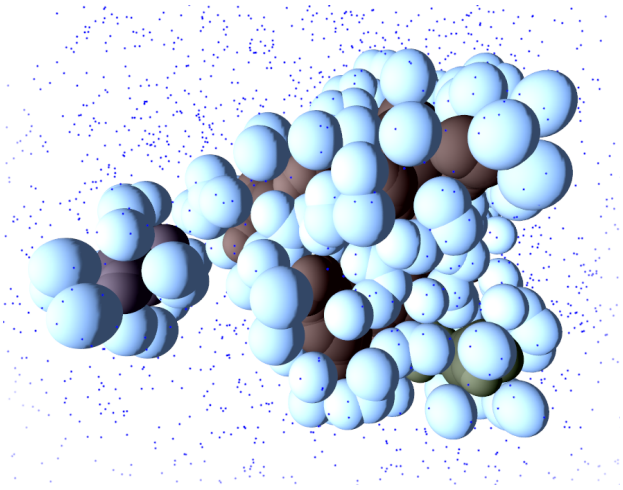
Probe Size [Å]	Time [ms]
0.2	174
0.4	146
0.6	114
0.8	91
1.0	75
1.2	64
1.4	55
1.6	47
1.8	41
2.0	35

Table 1: Dependency of the algorithm run time on the probe size.

Next, we have evaluated the run time of our algorithm on several molecules of different sizes. The measured



(a)



(b)

Figure 8: (a) Some of the cavities in the molecule 1AKD for the probe with the radius 1.4Å. (b) The cavities (dark) with the atoms forming them (light). Small dots represent centers of the molecule atoms.

times without preprocessing are presented in Table 2. In the first column, the PDB IDs of the tested molecules are presented. In the second column, the number of atoms is given. In the third column, the measured time is given. Figure 10 shows that the algorithm scales linearly for a variable molecule size.

Molecule	No. of atoms	Time [ms]
1CQW	2 754	3
3VMN	5 621	2
3AOB	23 385	14
3UXS	49 743	33
3OH5	87 539	55

Table 2: Algorithm run time for molecules of various sizes.

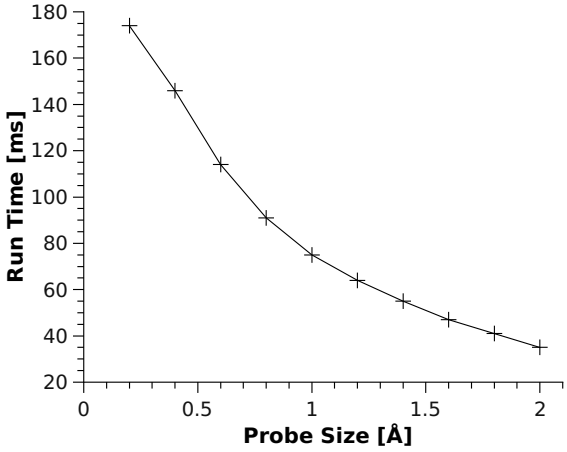


Figure 9: Dependency of the algorithm run time on the probe size.

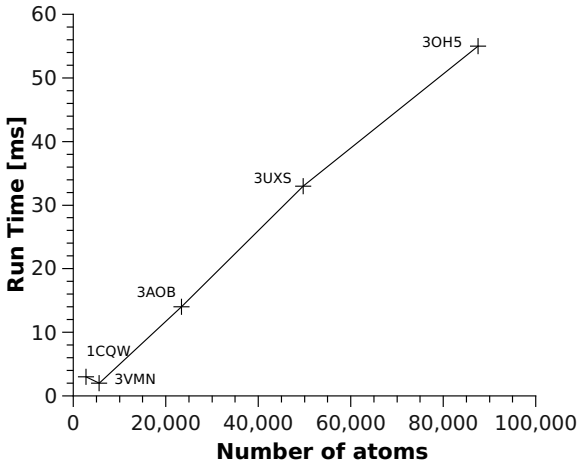


Figure 10: Dependency between the number of atoms and the run time for the probe with the radius of 1.4Å.

5.2 Comparison With Other Software

We have compared run time of our algorithm with Voroprot 0.7.6.4 [20], which implements similar algorithm using the additively weighted Voronoi diagram for finding cavities. We chose Voroprot also because we had problems running other implementations, such as CASTp [4] which is offered only as a web service.

Transaldolase from Corynebacterium glutamicum (PDB ID 3R5E, 2 957 atoms) was used for the comparison. The reason of the choice of such a small molecule was that we encountered the problems with processing larger molecules in Voroprot. The times for Voroprot are approximate, as the application has a graphical interface only. The measured times are in Table 3. It can be seen that our algorithm is several orders of magnitude faster than Voroprot. The reason is that Voroprot always searches all Voronoi vertices while our algorithm uses sorting to reduce the size of searched set.

Probe Size [Å]	Run Time [s]		Algorithm Speedup
	Our Algorithm	Voroprot	
0.2	7×10^{-3}	3.5	500
0.4	4×10^{-3}	2.5	625
0.6	3×10^{-3}	1.9	633
0.8	2×10^{-3}	1.4	700
1.0	2×10^{-3}	1.0	500

Table 3: Comparison with Voroprot.

5.3 Results, Summary and Future Work

We used the visualization tool to check the results visually. As far as we know, exact evaluation of results has not been developped yet. We have also compared the largest cavities found using our implementation with the cavities found using Voroprot.

Since Voroprot provides only a graphical interface, it was not possible to do exact run time measurements. Therefore, more exact time comparison should be carried out in the future.

Our algorithm currently does not handle elliptic edges. These edges may or may not be incident to any Voronoi vertex. It is also possible that only a part of an elliptic edge forms a cavity. Fortunately, elliptic edges are rare in proteins, because the difference among atom radii is small. This is left for future work.

6 Conclusion

We have presented the algorithm for finding inner cavities in a molecule. The algorithm computes the additively weighted Voronoi diagram of molecule atoms and sorts the Voronoi vertices using the distance to their generators in the preprocessing. The cavities are then found using a graph traversal, starting from the vertex with largest distance and ending when all cavities are found.

Our experiments shows that our algorithm is excellent for a variable probe size thanks to the preprocessing. The algorithm is able to process even large protein structures very quickly.

Acknowledgment

This work has been supported by the Grant Agency of the Czech Republic, project No. P202/10/1435 and project SGS-2010-028. The implementation uses the awVoronoi library [1], developed by Martin Maňák.

References

[1] http://awvoronoi.sourceforge.net/.

[2] P. Beneš, E. Chovancová, B. Kozlíková, A. Pavelka, O. Strnad, J. Brezovský, V. Šustr, M. Klvaňa, T. Szabó, A. Gora, M. Zamborský, L. Biedermannová, P. Medek, J. Damborský, and J. Sochor. Caver 2.1. http://www.caver.cz, 2009-2011.

[3] Y. Cho, D. Kim, H.-C. Lee, J. Y. Park, and D.-S. Kim. Reduction of the search space in the edge-tracing algorithm for the Voronoi diagram of 3D balls. In *ICCSA (1)*, pages 111–120, 2006.

[4] J. Dundas, Z. Ouyang, J. Tseng, A. Binkowski, Y. Turpaz, and J. Liang. CASTp: computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucleic Acids Research*, 34(suppl 2):W116–W118.

[5] H. Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13:415–440, Dec. 1995.

[6] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13:43–72, Jan. 1994.

[7] T. Exner, M. Keil, G. Moeckel, and J. Brickmann. Identifification of substrate channels and protein cavities. *Journal of Molecular Modeling*, 4(10):340–343, Oct. 1998.

[8] M. Hendlich, F. Rippmann, and G. Barnickel. Ligsite: automatic and efficient detection of potential small molecule-binding sites in proteins. *Journal of Molecular Graphics and Modelling*, 15(6):359–363, Dec. 1997.

[9] D. Kim, C.-H. Cho, Y. Cho, J. Ryu, J. Bhak, and D.-S. Kim. Pocket extraction on proteins via the voronoi diagram of spheres. *Journal of Molecular Graphics and Modelling*, 26(7):1104–1112, 2008.

[10] D.-S. Kim, Y. Cho, K. Sugihara, J. Ryu, and D. Kim. Three-dimensional beta-shapes and beta-complexes via quasi-triangulation. *Computer-Aided Design*, 42(10):911–929, 2010.

[11] D.-S. Kim, Y. Chob, and D. Kim. Euclidean voronoi diagram of 3d balls and its computation via tracing edges. *Computer-Aided Design*, 37:1412–1424, 2005.

[12] D.-S. Kim, D. Kim, Y. Cho, and K. Sugihara. Quasi-triangulation and interworld data structure in three dimensions. *Computer-Aided Design*, 38(7):808–819, 2006.

[13] D.-S. Kim, J.-K. Kim, C.-I. Won, C.-M. Kim, J. Y. Park, and J. Bhak. Sphericity of a protein via the β-complex. *Journal of Molecular Graphics and Modeling*, 28(7):636–649, 2010.

[14] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. *PROTEINS: Structure, Function, and Genetics*, 33(1):1–17, Oct. 1998.

[15] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: II. inaccessible cavities in proteins. *PROTEINS: Structure, Function, and Genetics*, 33(1):18–29, Oct. 1998.

[16] J. Liang, C. Woodward, and H. Edelsbrunner. Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design. *The Protein Society*, 7(9):1884–1897, Sept. 1998.

[17] M. Maňák and I. Kolingerová. Fast discovery of Voronoi vertices in the construction of Voronoi diagram of 3D balls. *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 95–104, 2010.

[18] N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, and M. L. Gavrilova. An algorithm for three-dimensional Voronoi S-network. *Journal of Computational Chemistry*, 27(14):1676–1692, 2006.

[19] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. John Wiley & Sons, Inc., 2nd edition, 2000.

[20] K. Olechnovič, M. Margelevičius, and Č. Venclovas. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics*, 27(5):723–724, 2010.

[21] S. Sonavane and P. Chakrabart. Cavities and atomic packing in protein structures and interfaces. *PLoS Computational Biology*, 4(9):e1000188, Sept. 2008.

[22] A. Tripathi and G. E. Kellogg. A novel and efficient tool for locating and characterizing protein cavities and binding sites. *Proteins: Structure, Function, and Bioinformatics*, 78(4):825–842, Mar. 2010.

[23] M. A. Williams, J. M. Goodfellow, and J. M. Thornton. Buried waters and internal cavities in monomeric proteins. *Protein Science*, 3(8):1224–1235, Aug. 1994.

[24] M. Zemek, M. Maňák, and I. Kolingerová. Advanced space filtering for the construction of 3D additively weighted Voronoi diagram. *ADVCOMP*, pages 37–43, 2011.

Applications & Image Recognition

Audio Guided Virtual Museums

Sanda Sljivo*
Supervised by: Selma Rizvic†

Faculty of Electrical Engineering Sarajevo
University of Sarajevo
Sarajevo / Bosnia and Herzegovina

Abstract

Nowadays virtual museums are implemented using variety of different concepts. Most of them contain storytelling in virtual environments. This paper introduces the concept of audio stories guiding the user through the virtual exhibition. Our goal is to explore if the audio storytelling can compensate movement limitations in a virtual environment. For evaluation of user feedback we use qualitative analysis methodology.

Keywords: storytelling, virtual museums, virtual environments

1 Introduction

Media globalization offers endless possibilities to visit and explore physically distant sites using virtual reality. Virtual museums (VM) are enabling the Internet users, as well as the real visitors on site, to access the exhibits, interact with them and learn about their background and context using computer graphics and multimedia. Recently, storytelling is becoming an important part of implementation of virtual museums, as it enhances the immersion of the visitor and upgrades the pure visual expression of the exhibits.

The goal of this paper is to discuss the recent developments in virtual museums concepts with various kinds of storytelling. Being a partner of Virtual Museum Transnational Network [4], some of our virtual museum projects are included in the EU funded research about the most immersive concept for the virtual museum of the future. After introduction of story guided virtual environments in [15, 14] we explore the concept of virtual museums guided only by audio stories, with very limited motion possibilities. The idea for introduction of this concept emerged after visiting the Anna Frank virtual museum [3]. While listening to the voice telling the story about the presented locations, we did not feel the need to move through the virtual environment, being immersed in the story itself. This inspired us to explore how the user would perceive the limitation of movement, being offered the audio story in the virtual museum.

*dandyrocket@gmail.com
†srizvic@etf.unsa.ba

The rest of the paper is organized as follows: Section 2 gives an overview of the related work in the field of virtual museums with storytelling; Section 3 presents the audio guided VM concept through the case study - virtual museum of Bosniak institute; Section 4 analyses the user feedback using quantitative analysis methodology and Section 5 offers our conclusions and directions for future work.

2 Related work

The use of storytelling technology in virtual museums is not new. Nowadays there are many virtual museums online that use digital stories in the presentation of their virtual exhibits. [13, 6, 8]. Different storytelling techniques are used, such as textual, audio, video or avatar-based storytelling. What is still quite absent, however, is the use of storytelling guidance through the virtual museum exhibitions as a whole - guidance that will provide visitors with an accurate and complete image of not just particular exhibit, but also events, moments or places in history, and help them to understand and appreciate the artefacts in their historical context. There are very few examples of such virtual museums online. They are implemented differently. In some of them visitors activate stories by walking through virtual environments (automatically or by pressing buttons). In other, visitors listen to story intros about virtual exhibitions prior to entering virtual environments. Virtual environments in those museums also vary from still 3D renders to panoramic 3D walk-through environments.

The National Palace Museum [11] has an exhibition hall guide for the antiquities in one of their permanent exhibition halls called The treasures of eight thousand years. The exhibition presents antiquities from different historical periods (starting from 6,200 B.C.) organized in a number of exhibition rooms. Each room has an audio guide and video story illustrated with old photographs and sometimes with 3D animations as well. The rooms are implemented as movable panoramic photographs of halls. The Virtual Smithsonian [1] allows visitors to take a virtual, audio guided, room-by-room tour of the whole museum. The visitor can navigate from a non-movable 3D hall en-

vironment to another one and explore hotspots which contain 3D artefacts, high resolution images, video and audio clips, etc.

In Anne Frank Secret Annex [2] virtual museum visitors are able to virtually walk and move around the panoramic 3D rendered photos of rooms in the house, in which the Frank family and other Jews lived and hid during the WWII. Aside to being able to explore the house in great detail and to click on various extras with additional information, such as text or films, visitors can hear the series of stories, accompanied with the ambient sound and music that explain what happened to the people in hiding. These audio narrations bring Anne Franks life story to people's attention all over the world. They are based on the stories from now famous Anne Frank's diary and reports of witnesses from the Anne Frank House archives.

In contrast to Anne Franks virtual museum where visitors play stories randomly by clicking on info nodes, Sarajevo Survival Tools [5] virtual museum uses linear storyline to share the story about the life of Sarajevo citizens who were forced to live under the siege for 3.5 years (1992-1996). To the best of our knowledge, it is the only known example of its kind. In this virtual museum visitors are fully guided through the virtual exhibitions of artefacts, manufactured by Sarajevans to survive the siege, by a linear, digital video story. The whole digital story is divided in a series of story segments organized in a logical sequence. Story segments are designed in such a way to present an interrelation of exhibits within a specific theme, all linked through a storyline. Each segment is played as intro story in prior to corresponding exhibition gallery. Exhibition galleries are implemented as non-movable virtual imaginary spaces with links to interactive 3D models of artefacts, movies about them and galleries of photos.

All these virtual museums use some kind of storytelling guidance through their collections. The question we would like to answer here is if it is possible, in those kinds of virtual museums, to use the story as the visual distraction so to make visitors of virtual environments do less moving and clicking and more listening and viewing when the story is interesting and compelling enough to distract their visual attention [9].

3 Audio guided Virtual Museum

In this paper we introduce the concept of audio stories guiding the user through the virtual exhibition. Our goal is to explore whether the audio storytelling can compensate movement limitations in a virtual environment. Our case study is the virtual museum of Bosniak Institute in Sarajevo [7].

The Bosniak Institute is a cultural centre located in Sarajevo, Bosnia and Herzegovina (Figure 1), focusing on promotion of the cultural heritage, historical truth and culture of the Bosniaks and the other nations with whom they have lived together for centuries. It was established by Adil Zulfikarpai. The institute is housed in a renovated sixteenth century Turkish bath and includes a library, an art centre, archive, collection of old manuscripts and old maps. There are also: a collection of Syrian furniture, a collection of furniture from Safvet Bey Baagi family (Bosnian writer considered to be the father of Bosnian Renaissance and one of Bosnia's most cherished poets at the turn of the 20th century), and also the collection of various items and furniture from Bosnian history and culture. The

Figure 1: Bosniak Institute: exterior (above) and interior (below)

virtual museum project was implemented through the lab coursework of the Computer Graphics course at the Faculty of Electrical Engineering in Sarajevo. Students were creating the individual exhibits virtual representations that were connected in the virtual museum through the joint virtual environment.

Creation of virtual environment (Figure 2) is done through the workflow displayed in Figure 3. We have taken pictures of the real museum and also of all the exhibits. Virtual environment is supposed to resemble a part of the real museum, and to be as realistic as possible. To make the virtual environment, we have used 3ds max and Flash. Exhibits are grouped in a similar way they are arranged in exhibition rooms in the real museum. For every virtual room a short audio story is recorded. The audio story is supposed to act as a curator in the real museum, to intrigue the visitor and to make him or her visit as much exhibits as possible. When the user visits the virtual museum, he or she is guided by audio stories. They should help the visitor in navigation through the virtual museum and introduce him/her with the context of the exhibition.

Figure 2: Home page of virtual museum

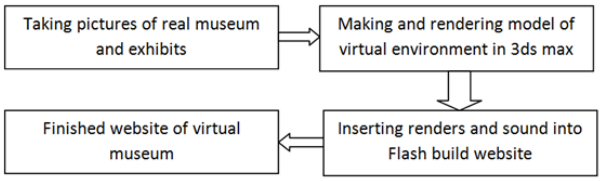


Figure 3: Workflow of VM creation process

Our virtual exhibition consists of the Syrian furniture, furniture from Safvet Bey Baagis house and collection of various cultural heritage objects. Structure of the VM is presented in Figure 4. The home page of the Virtual Museum is linked to the pages of: library, archive, manuscripts, maps, artwork benefactor, and 3D exhibits as presented in the Figure 2. When the home page loads, the audio story with information about Bosniak Institute starts. After the visitor clicks on the link to the 3D exhibitions, another page opens with audio story about the objects from the Bosniak Institute (Figure 5). From this page, visitor can select to go to one of three pages: Syrian furniture (Figure 6), furniture from Safvet Bey Basagics house (Figure 7) and the page with collection of various items and furniture (Figure 8). On every collections page, an audio story is also loaded with information about the selected collection. When the visitor clicks on a particular item from the selected collection, a new page opens with digital content related to that item (pictures, 3D model, video and gallery of photos, Figure 9). The visitor can mute the audio story in order not to be annoyed if he or she has already heard the story during the previous visits to the virtual museum.

4 Evaluation

Evaluation of the audio guided virtual museum concept has been conducted with two tools: questionnaires and in-depth interviews. The main goal of both user studies was to determine if the audio storytelling was enough for

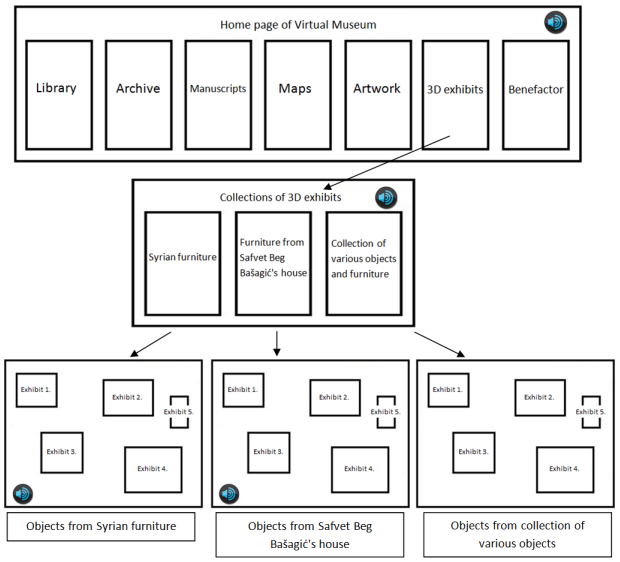


Figure 4: Structure of virtual environment

Figure 5: Selection of exhibit group

guiding the users through the VM collection, considering that they had no possibility to move in 3D virtual environment. The analysis of results was performed using qualitative analysis methodology [12] and therefore no statistical calculations were used for the analysis of results. Since the practice has shown that 7 users will find approximately 80% of problems is in graphics user interface [12], we have performed the user studies on 14 users in total.

4.1. Experiment design

4.1.1. User study based on questionnaires

Ten users participated in the study, 4 male and 6 female. They aged from 25 to 50, with the average age of 35. All of them reported normal hearing. Eight out of ten reported normal vision. There were no particular criteria for selection of users. All of them are from Bosnia and Herzegovina.

We have created a semi-structured questionnaire, which

Figure 6: Syrian furniture

Figure 7: Furniture that belonged to Safvet-Bey Basagic

includes both open-ended and specific questions (some of them displayed in Table 1), and sent it to the participants by email, along with the instruction document. They were asked to read the instructions, explore the virtual museum environment, fill in the questionnaire and send back their responses.

4.1.2. In-depth interviews

There were 4 users, 2 male and 2 female. They aged from 23 to 30, with an average age of 26. All of them reported normal hearing. Also, all of them reported normal vision. There were no specific criteria for selection of users and all of them were from Bosnia and Herzegovina. Users were interviewed based on the questionnaire from the previous section, but they had more freedom to express their opinions and time to discuss the topic.

4.2. Analysis of the results

Qualitative data analysis is based on data coding [12]. It is a process of extracting qualitative data into quantitative form. In such a process the possible values of the qualitative data are created according to the given answers. Since participants often use different terms for the same phenomenon or same words for different phenomena, it is important to perform coding as accurate as possible, without losing too much information.

Figure 8: Collection of various items and furniture

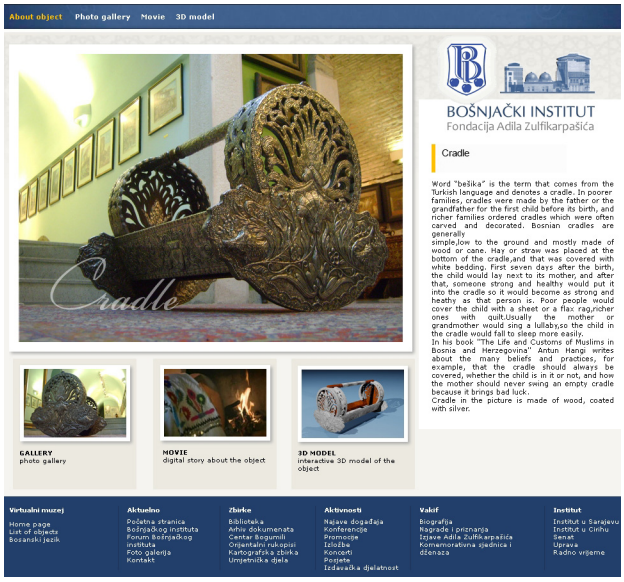


Figure 9: Cradle exhibit homepage

The data analysis was performed in two steps: defining the hypotheses and grounding the evidence. The hypotheses were generated using the constant comparison method [10]. After coding the questions (Table 2), each of them representing a particular section, we went through the data looking for patterns.

We built the following hypotheses from the data:

(H1) - audio story improves the quality of virtual museum presentation;

(H2) - having the story to guide them through the museum, users do not feel limited even if their movement in 3D environment is disabled. The aim of this study is not to prove our hypotheses, but to build up the weight of evidence supporting these propositions, that could be used as ground theories in future studies. Nine out of 14 users (10 from questionnaires + 4 in-depth interviews) were satisfied with the story. Two out of 14 users said that the story was good but it could be more dynamic and 2 out of 14 users said that story is too long, boring and distracting. Overall,

Question	Code	Possible value
What do you think about the story that guides through the virtual museum?	S1	Good Average Bad
Were you immersed in the story?	S2	Yes No
Did the story distract you from exploring the virtual museum?	S3	Yes No
Did the story contribute to your immersion in the environment?	S4	Yes No
What do you think about navigation in the virtual museum?	N1	Good Bad
Were you able to move in the 3D environment?	N2	Yes No
Did you feel the need for moving?	N3	Yes No

Table 1: The questions, codes and possible values used in the study.

Code	Answer
S1	Good(9) Average(2) Bad(2)
S2	Yes(11) No(2)
S3	Yes(1) No(12)
S4	Yes(9) No(4)
N1	Good(9) Bad(5)
N2	Yes(10) No(4)
N3	Yes(12) No(1)

Table 2: The codes and the number of answers provided.

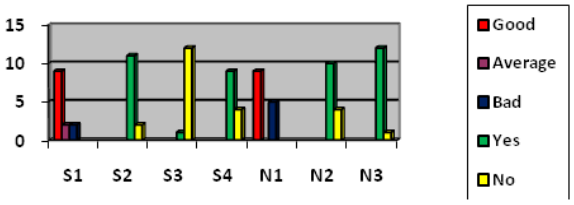


Figure 10: Graphic representation of answers

the users liked the story; it made them feel like they are in the real museum, they were immersed in the story itself. Also it was not distracting and it acted like a guide through the virtual museum. They argument their opinion on the story with the following explanations: We get two things in one - It can save us time and that is what we all want or Story gives the intimate touch to virtual environment; visitor has a feeling like he or she is not alone, he/she feels like somebody is guiding him/her. Results related to the codes S1-S4 give enough evidence to support our first hypothesis (Figure 10).

Nine out of 14 users liked the navigation - It is very intuitive, and Ive just followed one link to another. Five out of 14 users did not like the navigation; they found it as too static and had problems with returning to the previous pages (You have to click too much to go back on Home page of the Virtual Museum). One user did not answer some of the questions.

The most important result of the study is that 11 out of 14 users have not noticed that the movement in 3D environment was disabled (Figure 11). To the specific question Were you able to move in the 3D environment? they answered positively. Most probably some of them considered as movement changing of virtual environments using links, but however, they have not reported problems with lack of movement abilities. This result gives enough evidence to support our second hypothesis. Semi-structured questionnaires also allow collecting not only the foreseen information but some additional, unexpected data as well. Besides the mentioned, positive criticism of the project, some gaps of the proposed concept were identified. There were a few complaints and suggestions about the content: There could be some background music also, I know it is a student project, but could the sound be recorded by a professional narrator, Can pages be more interactive, as in a video game?. There were some suggestions related to the content of the story and design of the web site.

It was also interesting that users who are not from technical disciplines liked the project more than computer science professionals. Also, users who do not have much experience with virtual museums think that the environment is realistic, and do not have problems with navigation. Experienced computer users were more demanding in technical sense. They had more suggestions considering the navigation and technical realization. Almost all users said that they learned a lot from the project and would like to visit the real museum after visiting the virtual one.

5 Conclusions and future work

In the paper we presented the concept of audio guided virtual museum. This work is a part of our research on storytelling in virtual museum projects, performed inside the Virtual Museum Transnational Network. Results achieved so far show that the visitors appreciate story guided virtual museums, as they provide them with the context of the exhibition and historical background, not always visible from the virtual presentation of the very artefacts. The story also enhances their immersion in the different space and time.

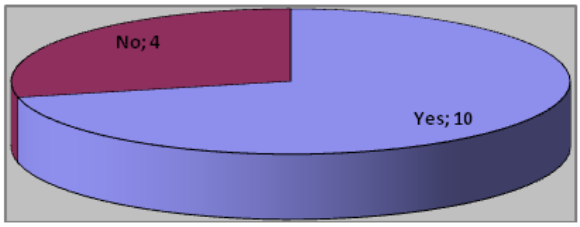


Figure 11: Graphic representation of answers on question N2

Audio guided virtual museum of Bosniak Institute introduced audio storytelling in virtual environments presented as still images with links (hot spots). We were interested to explore how the users react to the movement disability and if the audio story can make up that limitation without decreasing the overall quality of the virtual museum. Qualitative analysis of the user study results shows that most of the users have not even noticed that they are not able to move in the virtual environment. They also found that audio storytelling improves the overall quality of the virtual exhibition. In the future work we will perform more different user studies in order to find the best relationship between the storytelling and freedom of movement in virtual museum environments. We will also explore how to make the storytelling more interactive, incorporating it in serious games for cultural heritage.

6 Acknowledgements

Virtual museums developed by Sarajevo Graphics Group are included in FP7 NoE Virtual Museum Transnational Network V-MusT.net [4].

References

[1] G. Jones, M. Christal, The Future of Virtual Museums: On-Line, Immersive, 3D Environments, Austin, TX: Created realities group. http://createdrealities.com/virtual_museums.html/(accessed sep 2002), 2002.

[2] Anne Frank Museum Amsterdam. http://www.annefrank.org/(accessed feb 2012), 2012.

[3] Anne Franks 3D House. http://www.Annefrank.Org/En/Subsites/Home/Enter-The-3D-House/(accessed feb 2012), 2012.

[4] EU FP7 Network of Excellence Virtual Museums Transnational Network. http://www.v-must.net/(accessed feb 2012), 2012.

[5] Sarajevo Survival Tools. http://h.etf.unsa.ba/srp/(accessed feb 2012), 2012.

[6] Virtual Gallery of Rosicrucian Egyptian Museum. http://www.egyptianmuseum.org/virtualgallery/(accessed feb 2012), 2012.

[7] Virtual Museum of Bosniak Institute. http://h.etf.unsa.ba/bosnjacki-institut/index-eng.htm/(accessed mar 2012), 2012.

[8] Katie Simon Christopher Goodmaster Frederick Limp C. Scott Smallwood, Angelia Payne and Jackson Cothren. *Lighting Systems in Three Dimensional Non-Contact Digitizing: A View from the Virtual Hampson Museum Project*. CAA, 2009.

[9] Dadova Major Onacilova Sikudova Svarba Valikova Varhanikova Vataha Vesel Duskova Ferko, Cernekova. *Schola Ludus, Serious Games, and Measurement of Interestingness*. IEEE International Conference on Interactive Collaborative Learning (ICL), 2011.

[10] Strauss A. Glaser, B. *The discovery of grounded theory: Strategies for qualitative research*. Aldine Publ., 1977.

[11] Lin Quo-Ping Lai, Ting-Sheng. *The development of 'digital museum' and the innovative applications of e-learning at the national palace museum*. WSEAS Transactions on Computers Research. Vol. 1, no. 1, pp. 37-44, November 2006.

[12] Feng J.H. Hochheiser-H. Lazar, J. *Research Methods In Human-Computer Interaction*. Wiley, 2010.

[13] G. Scardozzi M. Cultraro, F. Gabellone. *The virtual musealization of archaeological sites: between documentation and communication*. ISPRS Proceedings, Volume XXXVIII-5/W1, 2009.

[14] V. Hulusic A. Karahasanovic S. Rizvic, A. Sadzak. *Interactive Digital Storytelling in the Sarajevo Survival Tools Virtual Environment*. SCCG, 2012.

[15] Selma Rizvic Vedad Hulusic. *Story Guided Virtual Environments in Educational Applications*. LNCS Transactions on Edutainment, 2012.

Simulation of Electronic Flight Instrument System of Boeing 787 aircraft

Andreas Stefanidis*
Supervised by: Miroslav Macik†

Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

Electronic Flight Instrument System (EFIS) known as glass cockpit is a key component of any modern aircraft. It visualises practically all cockpit instruments and serves as replacement for obsolete solution where most instruments were electromechanical. Flight crews must gain perfect knowledge of cockpit systems of each aircraft type in order to control it safely. For most aircraft types the available simulators are expensive, therefore difficult to access for most people. This paper describes realisation of a semi-professional simulator of Boeing 787 EFIS. Aim of the presented solution is to serve as simulator for demanding amateur users, for testing newly developed aircraft systems in the context of their future use and, in case of successful certification, also as a training device for flight crews. In this paper we present description of individual parts of the EFIS and discussion of currently available simulators for professional and amateur use. Furthermore, individual aspects of our solution are described in detail. Besides standard functions of EFIS it supports simulation of other aircraft systems, e.g. individual panels in the cockpit allowing interaction with the flight management system (FMS). Using a generic interface it allows integration with various systems for simulation of the external environment and physical model of the aircraft. Presented solution has been verified by functional testing and by specific usability tests with both professional and amateur users.

Keywords: Electronic Flight Instrument System, Flight Simulation, Flight Training, Human-computer Interaction, 2D Graphics

1 Introduction

Nowadays, the aviation is once again in the rise and global manufactures of civil aircrafts are introducing new variants of current aircraft models such as Boeing 747-8 or Boeing 737 MAX and even developing completely new models such as Airbus A350 XWB, A380 or Boeing 787. This trend also evokes increasing number of people interested in flying, not only as aircraft passengers, but also as those who like to try pilot experience.

After the events of 9/11 regulation in aviation security has tightened and access to the flight deck has become almost impossible for regular people. Flight simulators, either professional or amateur, provide more or less real flight experience, which solves this issue to some extend. Professional simulators usually provide experience close to a real flight, but access to them can be very costly. The other option is usage of amateur simulators, which can be used in a home environment for an affordable price. These simulators are usually not even close to the quality level of the professional devices. Mostly, they are product of the entertainment industry, such as PC games, which are mainly aimed on amateur users. Therefore they do not provide complex simulation.

The Boeing 787 is same as the vast majority of current airliners equipped with a glass cockpit – a cockpit consisting of LCD panels instead of electromechanical devices. This system is called Electronic Flight Instrument System (EFIS). Main goal of this work is to implement a simulation of Boeing 787 EFIS, which would allow all users, even those who do not have access to professional simulators, use a system providing highest possible quality of simulation of all systems in the cockpit.

This work focuses on following areas:

- On group of people who are interested in aviation and would like to experience control of an airliner but professional simulators are unavailable for them. Quality of home simulators is not sufficient for them, and thus they are seeking for a product which would provide, in home conditions, simulation comparable to professional solutions for an affordable price.
- To provide environment for testing of aircraft systems being developed in context of their future use.
- To serve as certified flight training device – Basic Instrument Training Device (BITD, see further) and/or Flight and Navigation Procedures Trainer (FNPT, see further).

*stefaan1@fel.cvut.cz
†macikmir@fel.cvut.cz

2 Background

This chapter introduces the Boeing 787 Dreamliner and basic terms from the field of aviation instrumentation necessary for comprehension of this text.

2.1 Boeing 787

Boeing 787 Dreamliner is currently the last model of civil aircraft from the Boeing Corporation. It is a twin-engine aircraft, which should eventually replace the Boeing 767 model. It is the first civil aircraft that use composite materials instead of aluminium in such a large scale. Furthermore, thanks to a new design of engines, the noise produced by the plane is decreased by 60% and the fuel consumption by 20% respectively [18]. Modern construction of this aircraft incorporates modern design of the EFIS in ins cockpit.

2.2 Electronic Flight Instrument System

In this work we focus mainly on the simulation of LCD panels which are present in an aircraft cockpit. The main part of the Boeing 787 glass cockpit consists of five fifteen-inch LCD panels. These LCD panels are then virtually divided into smaller parts (see Figure 1):

- Primary Flight Display (PFD) – Display that provides pilots with the most important flight data like airspeed, altitude, attitude, heading etc.
- Multi-Function Display (MFD) – MFD can display various panels according to data that are required at a particular situation. Specifically it can display navigation display (ND), system display (SYS), electronic checklist (CHKL), Control Display Unit (CDU), Information Display (INFO) and Communication display (COMM).
- Engine Indicating and Crew Alerting System (EICAS) – Provides flight crew with information about engines condition and also displays annunciations for the crew.

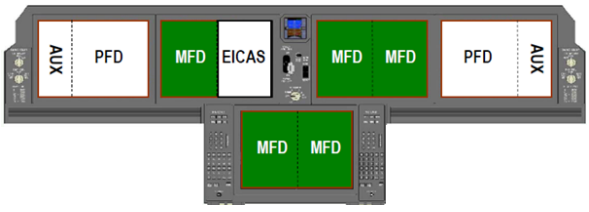


Figure 1: EFIS parts [1]

The cockpit is also equipped with Integrated Standby Flight Display (ISFD), Head-Up Displays (HUD) and Electronic Flight Bags (EFB). Normally, these systems are not vital for successful flight and description of these systems is out of scope of this work (however detailed information can be found in[14]).

2.3 Flight simulators

Flight simulators can be divided into two groups – amateur and professional. The former group incorporates mainly PC games, which allows users to experience the role of a pilot with no need for expensive flight training. This category is represented for example by Microsoft Flight Simulator [9] or XPlane [8]. The latter group represents professional simulators that are used for training of flight crews and therefore must satisfy strict requirements for accuracy of the simulation. Based on the level of the simulation, there are four main categories of professional flight simulators:

- Basic Instrument Training Device (BITD) – A ground based training device. It may use screen based instrument panels and spring-loaded flight controls, providing a training platform for at least the procedural aspects of instrument flight [7].
- Flight and Navigation Procedures Trainer (FNPT) – A training device which represents the flight deck or cockpit environment including the assemblage of equipment and computer programmes necessary to represent an aircraft in flight operations to the extent that the systems appear to function as in an aircraft [7].
- Flight Training Device (FTD) – A full size replica of a specific aircraft types instruments, equipment, panels and controls in an open flight deck area or an enclosed aircraft flight deck, including the assemblage of equipment and computer software programmes necessary to represent the aircraft in ground and flight conditions to the extent of the systems installed in the device [7].
- Full Flight Simulator (FFS) – A full size replica of a specific type or make, model and series aircraft flight deck, including the assemblage of all equipment and computer programmes necessary to represent the aircraft in ground and flight operations, a visual system providing an out of the flight deck view, and a force cueing motion system [7].

Created EFIS should meet the requirements for professional simulator from category BITD and/or FNPT, because for this category is not required complete simulation of all aircraft systems.

3 Related work

This chapter provides a brief review of currently available solutions providing relevant functionality.

Abacus 787 is a plug-in module for Microsoft Flight Simulator X that provides simulation of Boeing 787 aircraft. Although, this system provides visualisation of EFIS visually close to reality, the functionality is not optimal (see Figure 2). The panels show only fraction of the data needed in a real flight and the Flight Management Computer (FMC), which is a vital component, is missing completely. This prevents the system from providing realistic experience [17].

Figure 2: Abacus 787 [17]

Project magenta (PM) is EFIS simulator of Boeing 737NG, 747-400, 777 and 757/767 or Airbus A320, 330 and 340. Project Magenta provides not only simulation of the EFIS (see Figure 3) but also many other systems such as control panels or an instructor station. Most current amateur cockpit simulators are based on this project. However, it does not provide simulation of the Boeing 787 EFIS. It is not an open project, therefore it is not possible to extend it in order to simulate B787 EFIS.

Figure 3: Project Magenta [4]

Thales 787 is a professional simulator of Boeing 787 certified as category Level D from Thales Group (see Figure 4). Because its price around USD 15-20 million it is not available to most regular users.

Figure 4: Thales 787 [5]

4 EFIS simulator architecture

This section describes the architecture of the created simulator and some specific issues of the implementation.

The implementation is based on C++, OpenGL for graphical output and XAudio2 for audio output respectively. For communication with the Windows OS it uses WinAPI interface.

A modular design has been used in order to achieve an universal solution, where individual independent components are created as DLL libraries. The systems consist of three basic composes:

- The program core, which provides simulation of the EFIS itself.
- Module for communication with simulation platform that provides simulation of the external environment and physical model of the aircraft. The system can be extended in order to support another simulation platform by using appropriate version of this module.
- Module for communication with peripheral devices such as joystick, physical panels etc.

Because the program consists of several modules that work independently, it was necessary to ensure that these parts do not slow down each other, especially parts operating in precise time intervals. We used multi-threaded design to satisfy this requirement (see Figure 5).

EFIS essentially requires multiple displays, in our case five. Therefore it is necessary to allow our system to be displayed on multiple monitors as well. Graphic cards that support connection of more than two monitors at same time are still not very common. Therefore it was necessary to design the system to support multiple graphic cards at the same time.

On each monitor the content is displayed in an independent window, which guarantees hardware acceleration for each display. Therefore it was necessary to design rendering loop to be able to render into multiple windows simultaneously and work effectively as well. Our solution for

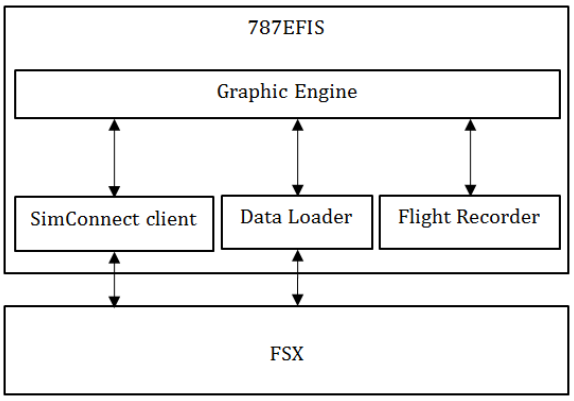


Figure 5: Structure of threads

this is to use an independent thread for each monitor (see Figure 6). All these threads contain an independent rendering loop. If we have more CPUs and/or GPUs this solution is significantly more efficient than a naive method (rendering all windows in one thread) [3]. On the other hand it requires synchronisation between threads and data sharing between different graphic cards. All created threads have access to shared memory where all necessary information about the current simulation state is stored.

It is also possible for users to choose on which monitor and position will be the individual parts of EFIS displayed. This customisation is possible by using configuration file.

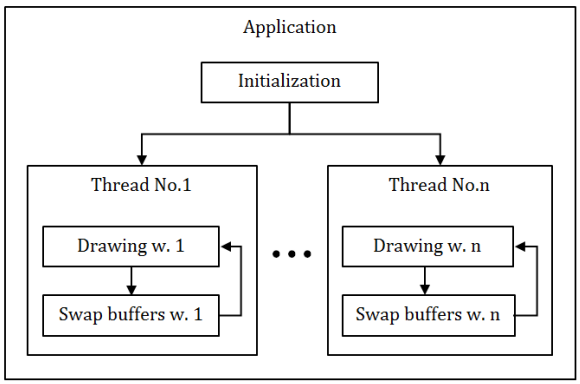


Figure 6: Rendering with multiple threads

Besides the rendering loop, any Windows® application contains its own loop for receiving messages through which it reacts to external events such as system messages, status information about mouse and keyboard etc. Due to the fact that our program contains multiple windows, it was necessary to adapt message loops of particular windows. Consequentially it was necessary to separate messages on those important for each application window and those important for the entire program (see Figure 7).

In order to allow interaction with the created EFIS our solution also contains simulation of some control panels from the cockpit. For their simulation we created a custom

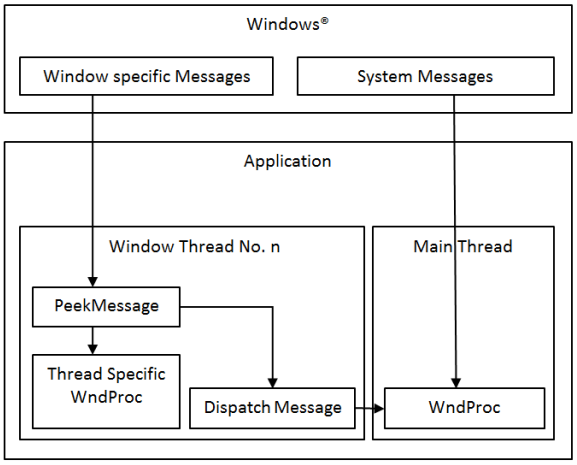


Figure 7: Processing messages

system. It is able to display these panels and react to events trigged by the user as well.

Another important component of the created simulation is Flight Management Computer (FMC). FMC contains three databases:

- Navigation database – contains informations about all airports and all navigation points.
- Performance database – contains performance characteristics of the aircraft.
- Airline modifiable information – contains airline specific data.

FMC is then able to calculate and provide important informations to crew and other systems. For example informations about minimum and maximum speed at particular situation or navigation information displayed on Navigation Display (ND) or Control Display Unit (CDU).

5 External simulation platform interface

Our system serves as simulation of the EFIS. In order to achieve complex simulation, it was necessary to integrate it with a platform that provides simulation of an external environment and physical model of the aircraft. Microsoft Flight Simulator X (FSX) meets most of our requirements, therefore we used this platform [3].

Various data about the external environment and the aircraft physical model are required for operation of the presented EFIS simulator. These data can be divided into three categories:

- Data whose accuracy is critical and hence require frequent updates. This category includes data like aircraft position, pitch, bank etc.

- Data that are not changing rapidly during the simulation and therefore does not require frequent updates. This group includes for example weather information or position of other aircrafts in a particular area.

- Data constant during the simulation, for example navigational information.

It was necessary to take into account these requirements when designing the communication interface between our simulator and FSX. To obtain time-critical data we used SimConnect [10] interface. This interface is part of the Microsoft FSX. It allows communication between FSX and other modules. SimConnect works on client-server principle. Information about the state of the simulated aircraft such as position, altitude etc. can be easily obtained by sending requests via the SimConnect interface into FSX [11]. This principle cannot be applied to all requirements of our system, because not all required data can be obtained using the SimConnects in this straightforward way (e.g. data for systems Traffic Collision Avoidance System (TCAS), Vertical Situation Display (VSD) or terrain radar). Further text describes solutions how to obtain required data that are not directly accessible using SimConnect interface.

For implementation of the TCAS it is necessary to get information about position of other aircrafts in the area. These data can be obtained via SimConnect using method as described in [6]. In the beginning it is sent request to FSX for list of all objects in the simulation. After that is sent for each object request for its actual position and altitude. These data are then requested periodically(see Figure 8). Also it is necessary to watch events when some object enter or leave simulation.

For the VSD, it is necessary to get data about terrain in front of aircraft. It is not possible to obtain data about the terrain using the SimConnect interface. However we can use similar principle which is used to simulate the TCAS. In our program we will first insert an object into FSX. This will serve as an invisible probe. Position of the probe will be then incrementally moved in the axis of the aircraft and at every change of the position we get height of the terrain (see Figure 9). Results of this solution are good and situation on VSD corresponds with relief in the FSX. However, because implementation of sampling does not use any buffer to store and/or preload the necessary data, after change of heading, a resampling of the new relief on the VSD is visible.

To obtain the static data, we decided to use decompilation [21] of FSX data files by using modified version of decompiler BGL2XML [2].

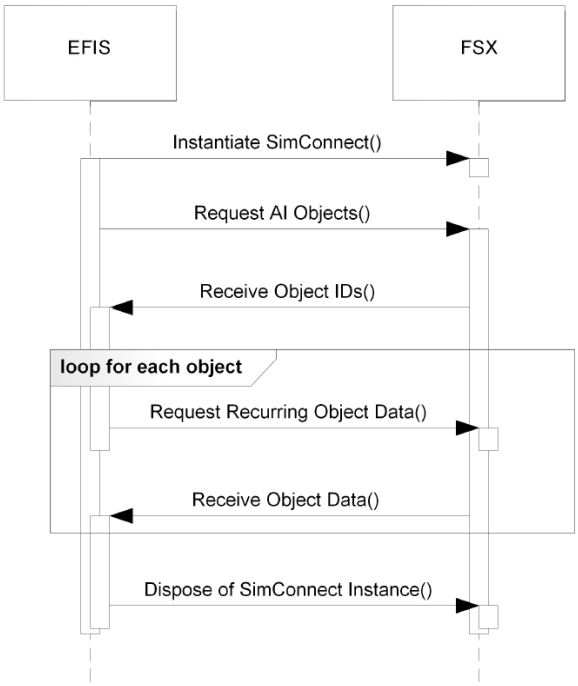


Figure 8: TCAS

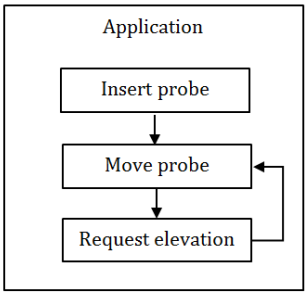


Figure 9: Using probe for getting terrain profile

6 Results and Evaluation

Created system fully implements following EFIS subsystems:

- Primary Flight Display (PFD)
- Navigation Display (ND)
- Engines Information and Crew Alerting System (EICAS)
- Traffic Alert and Collision Avoidance System (TCAS)
- Control Display Unit (CDU)
- Vertical Situation Display (VSD)
- Control panels: Mode Control Panel (MCP), EFIS Control Panel (ECP), Display Selection Panel (DSP), Multi-functional Keypad (MFK), Glareshield panel (GSP)

Furthermore some systems are supported partially, namely: FMC (Flight Management Computer), Synpop-

tics display (SYS), Auxiliary display (AUX), Centre Forward Panel.

Presented system has been tested for functional and performance aspects. Functional testing was performed by testing with users and by comparison with real system. For performance testing we used standard methods such as unit testing and profiling.

6.1 User interface testing

Because our system is a very specific product, testing of its user interface requires users with at least basic aviation experience. The testing was performed in a manner of an expert review with users who are familiar with function of an aircraft EFIS. Testing itself was made with two different user groups, those who gained their experience with EFIS in simulators and those with experience from a real aircraft. This way we get feedback from people with different perspective. The test also covered intended target categories of users.

The test was performed on a system consisting of two PCs. The first one was running Microsoft Flight Simulator X with one monitor. The other PC was running our EFIS simulator and was equipped with two monitors. Complete testing layout is shown in Figure 10. The system was controlled by a mouse and Saitek X52 Pro joystick with pedals.

Figure 10: Testing layout

The testing itself was divided into several parts:

- Introduction of the project
- Introducing the EFIS, its possibilities, limitations and the way of control
- Pre-test interview
- The actual test. The test consisted of several tasks, each of them aimed to verify a particular part of created system
- Semi-structured post-test interview

Overall assessment from the users was positive. However performed tests revealed certain factors, where it is necessary to improve the created system:

- Hardware problems - Due to hardware limitations, the layout used for testing was not same as layout in a real aircraft. This caused problems during interaction with the system.
- Problems caused by incomplete implementation - Because the created EFIS does not simulate all aircraft systems, it was not possible to test all sub-systems, especially the FMC.
- Problems caused by mistakes in implementation discovered during testing. Some of those issues were already fixed.

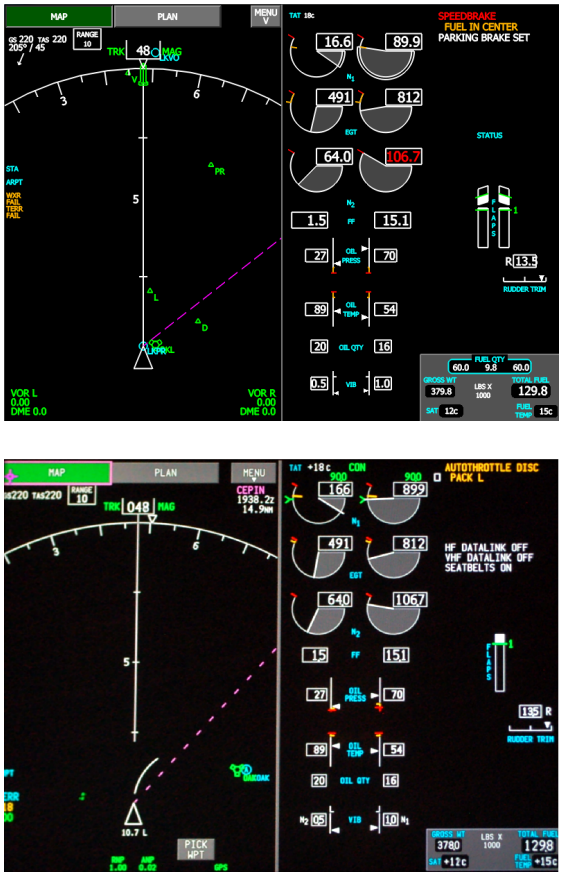


Figure 11: Comparison with the real system (top - our simulator, bottom - real system [5])

6.2 Comparison with the real system

It was practically impossible to compare our system directly with the real system in Boeing 787 or with higher category simulator. Also it was not possible to invite users with experience directly from Boeing 787. Therefore we

compared the general functionality of individual systems with other planes. This allowed us to test the system by the functional aspect, because the basic function of the systems in different aircraft types is similar[20]. However comparison with the specific systems of B787 can only be obtained from available sources such as manuals for aircraft [14, 16, 15, 13], photos or videos. In Figure 11 and Figure 12 is a visual comparison of our system with photography of the real system. These figures depicts high fidelity of the simulated system. Most differences between these images are caused by slightly different state between simulated EFIS and the real system on the photography.

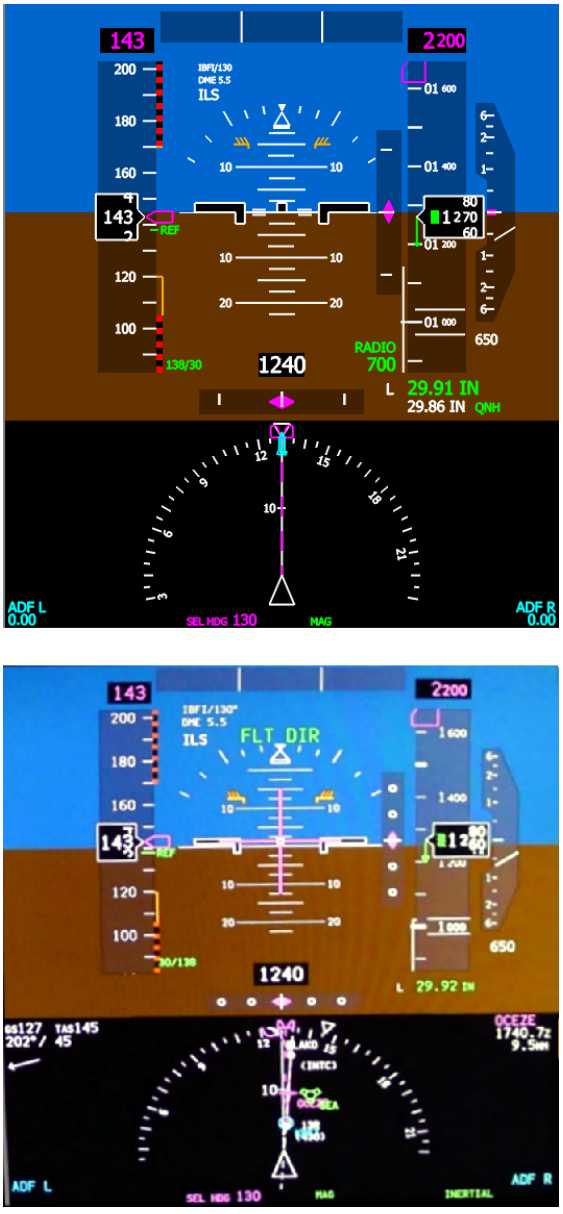


Figure 12: Comparison with the real system (top - our simulator, bottom - real system [5])

6.3 Unit testing and performance evaluation

A set of unit tests was performed in order to evaluate our implementation. Following subsystems were tested:

- Navigation database
- Thread management
- Interface between EFIS and FSX

Created system passed all performed unit tests. It is out of scope of this work to describe the results in detail, however the more information can be found here[3]. Moreover, the performance of the system was evaluated using profiling. Test itself was performed using integrated profiler in MS Visual Studio. During simulator testing we were running EFIS simulation on standard conditions for the period of one minute. From profiling results we concluded some recommendations to improve our system performance. The most significant one was to reduce data transfers between RAM and GPU. For example by using Vertex Buffer Objects or Vertex Arrays. Second recommendation was to improve synchronization between individual threads. The more detailed information and results can be found in [3].

7 Conclusions and Future Work

In this paper we presented design, implementation, evaluation and possible usage of our Boeing 787 EFIS simulator. Our implementation of EFIS could be used in the following scenarios:

- As a sophisticated EFIS simulator for advanced amateur users requiring high fidelity simulation.
- Thanks to its modular design it can serve as environment for laboratory testing of aircraft systems/instruments being developed or for testing interaction between cockpit systems and the flight crew.
- In case of successful certification from Federal Aviation Administration (FAA) [19] or European Aviation Safety Agency (EASA) [7] (or other national aviation authority) it could be used as a professional simulator.

Evaluation of our implementation of Boeing 787 EFIS proved that it provides functional simulation of a real system. We managed to design and implement quality simulation system, which includes most functions of the real EFIS and other aircraft systems.

Presented solution supports all basic function necessary for realistic simulation of an EFIS however the overall experience can be improved by finalizing some missing parts, namely: missing FMC functions, Synoptics display (screens representing status of hydraulic and electrical systems), Electronic Checklist and Head-up display.

Comparison with the real system has been performed using publicly available resources, however precise technical data are necessary for certification of the system as a professional simulator. This certification requires implementation of requirements described in Qualification Test Guide (QTG)[7]. Satisfying these requirements is then costly and may be difficult without information provided by aircraft manufacture.

Our system uses a modular design, therefore it is quite simple to enable support of another aircraft EFIS. There is already demand for EFIS simulator of Boeing 737 MAX. There is also demand for implementation of interface to the Flight Gear simulator – an open source project often used by the research community, e.g. in project [12].

8 Acknowledgements

This research has been done within project Automatically generated user interfaces in nomadic applications, funded by grant no. SGS10/290/OHK3/3T/13 (FIS 10-802900).

References

[1] Alan R. Jacobsen. 787 Flight Deck Displays, Boeing Commercial Airplanes, 2006.

[2] Alessandro G. Antonini. BGL to XML converter, Central Park Informatica, 2004.

[3] Andreas Stefanidis. Simulace systemu EFIS letounu Boeing 787, Czech technical university in Prague [online] https://sdip.felk.cvut.cz /browse/pdfcache/stefaan1_2012dipl.pdf. Master's thesis, 2012.

[4] Enrico Schiratti. Project Magenta [online] http://www.projectmagenta.com/, Accessed 08/11 2011, 2011.

[5] Jason Paur. Try Before You Fly: How Dreamliner Pilots Train Without Lifting Off [online] http://www.wired.com/autopia/2011/09/787-flight-training/, Accessed 11/13 2011 , 2011.

[6] Jessica Zorich. Monitoring AI Objects, Microsoft Corporation, 2008.

[7] Joint Aviation Authorities Training Organisation. JAR-FSTD A - Aeroplane Flight Simulation Training Devices, 2008.

[8] Laminar Research. XPlane, [online] www.x-plane.com/, Accessed 08/22 2011, 2011.

[9] Microsoft Corporation. Microsoft Flight Simulator X, [online] www.microsoft.com/games/flightsimulatorx/, Accessed 08/22 2011, 2008.

[10] Microsoft Corporation. SimConnect SDK Reference [online] http://msdn.microsoft.com/en-us/library/cc526983.aspx, Accessed 11/8 2011, 2008.

[11] Microsoft Corporation. Simulation Variables [online] http://msdn.microsoft.com/en-us/library/cc526981.aspx, Accessed 11/08 2011, 2008.

[12] Sipos, M. and Paces, P. and Reinstein, M. and Rohac, J. Flight attitude track reconstruction using two AHRS units under laboratory condition, Sensors, 2009 IEEE oct. 2009 675-678 ISSN=1930-0395.

[13] The Boeing Company. 777/787 Flight Crew Training Manual, Document Number FCT 777/787, Revision Number: 2, 2010.

[14] The Boeing Company. 787-8 Flight Crew Operations Manual, Document Number D615Z003-TBC, Revision Number: 4, 2010.

[15] The Boeing Company. 787 Flight Crew Training Manual, Document Number FCT 787, Revision Number: 2, 2010.

[16] The Boeing Company. 787 Quick Reference Handbook, Document Number D615Z003-TBC, 2010.

[17] Tim Capps. AVSIM Commercial Aircraft Review, [online] http://www.avsim.com/pages/0909/Abacus/787.htm, Accessed 08/23 2011, 2009.

[18] Tim Nelson. 787 Systems and Performance, Boeing Commercial Airplanes, Flight Operations Engineering, 2005.

[19] US Federal Aviation Administration. Federal Register / Vol. 71, No. 209 , 14 CFR Part 60 , [online] http://www.faa.gov /about/initiatives/nsp/media/consolidated_version.pdf, 2008.

[20] Wikipedia. ARINC 600 Series, [online] http://en.wikipedia.org/wiki/ARINC, Accessed 1/12 2012, 2012.

[21] Winfried Orthmann. FS X BGL File structure, [online] http://www.fsdeveloper.com/forum/showthread.php?t=3710, Accessed 9/13 2011, 2006.

Semantic Categorization and Retrieval of Natural Scene Images

Kristína Lidayová*
Supervised by: RNDr. Elena Šikudová, PhD.†

Faculty of Mathematics, Physics, and Informatics
Comenius University
Bratislava / Slovakia

Abstract

The semantic gap between the digital image representation and the user's image understanding is still a big problem. In our work we try to reduce this semantic gap in a field of natural images.

This paper proposes a method for semantic categorization and retrieval of natural scene images with and without people. These are typical holiday pictures from hiking outdoors. Our approach comprises of three stages. Preprocessing consists of image segmentation into arbitrary-shaped regions and detection of people in the image. In the next stage, local image regions are classified using low level features into semantic concept classes such as water, sky or sand. Finally the frequency of occurrence of these semantic concept classes determines the high level scene category. For the classification of local image regions the k-Nearest Neighbor and Support Vector Machine classifiers are used. The results obtained by both classifiers are validated within the paper.

Keywords: Semantic gap, Semantic retrieval, Content Based Image Retrieval (CBIR), Classification

1 Introduction

We live in a world where having a digital camera or image scanner is not a problem any more. People are used to take thousands of pictures during their vacation and they like to share them at the web galleries or social networks. Due to more and more images being generated in digital form around the world, it is important to deal with a problem how to extract the semantic content of images and then retrieve these images effectively.

Humans tend to interpret images using high-level concepts, they are able to identify keywords, abstract objects or events presented in the image. However, for a computer the image content is a matrix of pixels, which can be summarized by low-level color, texture or shape features. The lack of correlation between the high-level concepts that a user requires and the low-level features that image retrieval systems offer is the semantic gap.

Figure 1: An example of the semantic gap problem. The two images possess very similar colour and position characteristics, but differ vastly as far as the semantics are concerned.

In our work we try to reduce this semantic gap in a field of natural scene images with and without people. These sort of pictures are common in personal family albums. Our method can help the people to search in these albums effectively.

This paper is organized in the following way: The techniques in reducing the semantic gap are discussed in Section 2. In Section 3 our method for semantic categorization and retrieval is presented. In Section 4 we describe segmentation algorithm and body detection used here. Section 5 is dedicated for the low-level features and classifiers. In section 6 we deal with scene categorization and Section 7 discuss the results. Finally, Section 8 concludes the paper.

2 Related work

Image retrieval research is moving from text-based, through content-based, towards semantic-based image retrieval. Several systems reducing the semantic gap have been proposed.

In [6] the techniques reducing the semantic gap are divided into six categories:

1. **Object ontology** This system is using object ontology to define high-level concepts. Firstly low-level features describing the color, position and shape of each

*lidayova@gmail.com
†sikudova@sccg.sk

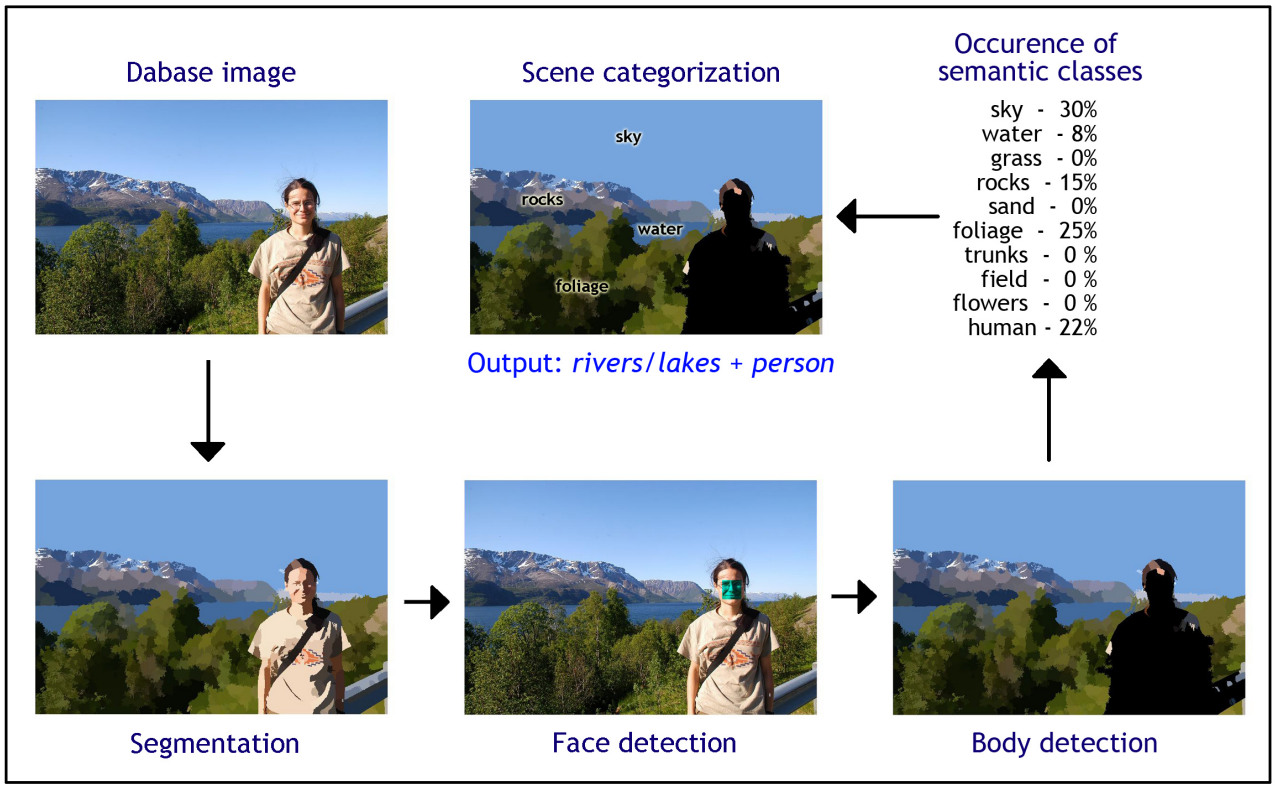


Figure 2: Overview of proposed method

region are calculated. Next different intervals for these features are defined. Each interval can be translated to an intermediate-level descriptor qualitatively describing the region attribute, that humans are more familiar with. These descriptors form a simple vocabulary, the so-called object ontology. Images can be classified into different categories by mapping such descriptors to high-level semantics based on our knowledge. For example "sky" can be defined as region of "light blue" color and "upper" spatial location. A typical example of such ontology-based system is presented in Ref. [8]

2. **Machine learning** This technique is based on using supervised or unsupervised machine learning tools to associate low-level features with query concepts. A supervised learning algorithm analyzes the training data and produces an inferred function, which should predict the correct output value for any valid input data. In unsupervised learning the goal is to describe how the unlabeled input data are organized or clustered. A novel scheme that combines semi-supervised learning, ensemble learning and active learning in a uniform framework is proposed in Ref. [13]

3. **Relevance feedback** Methods using relevance feedback technique work on-line and try to learn the user's intentions on the fly. At the beginning system provides initial retrieval results and the user marks which images he considers as "relevant" and which as "irrelevant". Machine learning algorithm learns the user's feedback and the selector retrieves another images. The process is repeated until the user is satisfied with the results. Mechanism of relevance feedback is well used in Ref. [7]

4. **Semantic template** This technique is not so widely used. Semantic templates are generating to support high-level image retrieval. Semantic template is usually defined as the "representative" feature of concept calculated from a collection of sample images. This technique is used in Ref. [14]

5. **Web image retrieval** This system has some technical difference from image retrieval in other application. Some additional information like the URL of image file or the descriptive text surrounding the image can help the semantic-based image retrieval.

6. **Frequency domain features** Image search and retrieval in this method mainly focuses on feature vectors based on the real and imaginary parts of the complex numbers of the image transformed by the Fast Fourier transform (FFT), Discrete Cosine transform (DCT) or WALSH transform. This technique was recently presented in Ref. [5]

Many systems exploit one or more of the above techniques to implement high-level semantic-based image retrieval. Our system consists of both supervised and unsupervised machine learning technique and semantic template for scene categorization.

3 Proposed method

Our work is based on the work [12]. Like in their method also in our proposed method the image is segmented into local subregions. The difference is in the shape of local image subregions. While in the initial method [12] the local image subregions are extracted on a regular grid of 10x10 regions, our proposed method tries to segment the image into arbitrary-shaped subregions, which correspond to objects boundaries. This improvement reduces the misclassification of regular subregions belonging to two or even more semantic concepts.

In addition our proposed method detects presence of people in the image. This is useful because our target images are typical holiday pictures from hiking outdoors. Presence of family members on this kind of images is very common, so it is important to cover also this condition into image retrieval process. So in our system it is possible to define if the retrieval pictures should contain people or not.

Only local subregion that represent nature are further processed. Thus we identify subregions belonging to people and separate them from others. Afterwards using low-level features we classify each subregion into one of following semantic concepts: *sky, water, grass, trunks, foliage, rocks, flowers and sand*. The selection of these local semantic concepts was influenced by the psychophysical studies of Mojsilovic et al. [9] and by concepts used in Ref. [12]. For the classification of local image regions we involved the k-Nearest Neighbor and Support Vector Machine classifiers.

The last stage of this proposed method is scene categorization. In our work we have six different scene categories: *coasts, forests, rivers/lakes, sky/clouds, plains and mountains*. To each local semantic concept, its frequency of occurrence is determined. This information enables us to make a global statement about the amount of particular concept being present in the image e.g. "There is 22% of water in the image." Using this knowledge the most suitable category prototype is assigned to the image, that gives the semantic meaning of the image.

4 Preprocessing

Preprocessing consists of image segmentation into arbitrary-shaped regions and detection of people in the image.

(a) (b)

Figure 3: An example of the body detection. (a) Results from the face detector with templates (b) Filtered out subregions belonging to humans bodies

4.1 Image segmentation

At first step of our algorithm the image is segmented into arbitrary-shaped subregions. We take advantage of Mean Shift segmentation algorithm presented in [3] based on grouping pixels, which are close in the spatial and color range domain. This algorithm iteratively detects modes in a probability density function.

It starts with a region of interest where kernel function calculates the mean shift vector. Using this vector the region is shifted to the new location. Can be shown that the mean shift vector is proportional to the normalized density gradient estimation, so the region certainly converges to a point with zero gradient. This is the mode corresponding to the initial position. Modes that are close to each other are grouped together. For segmentation purposes, each pixel is marked by color value of the corresponding mode.

4.2 Body detection

After the image is segmented we used algorithms for skin and face detection to identify subregions belonging to the humans body. We applied skin detection [10] which results in skin probability maps. For face detection we used the implementation of Viola/Jones Face Detector [11] found in [4]. This face detector is applied only in the regions, where skin was detected. This combination with skin detector produces more precise results, because occurrence of false faces in treetops and rocks was eliminated.

As next step a template in the shape of humans body is added to each detected face. Each subregion overlaid by this template is examined if the majority of subregion area lies inside the template or outside. Subregions lying for the most part within the template we consider to depict the humans body. They will not proceed to further processing and classification.

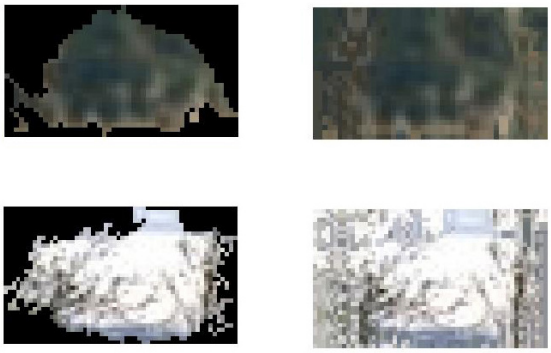


Figure 4: Example of extended subregions

5 Semantic Concept Classification

In the second stage three kinds of features are extracted from each subregion. Afterwards the subregions are classified by k-Nearest Neighbor and Support Vector Machine classifiers into eight semantic concept classes.

5.1 Color features

The color feature is one of the most widely used visual features in the image retrieval. In our work we use linear $L^*a^*b^*$ color histogram. L^* represents the lightness, a^* the red-green component and b^* the blue-yellow component. Colour histogram describes the distribution of color and lightness within the subregions. The histogram is invariant to rotation, translation and scaling, but does not contain semantic information.

5.2 Edge direction features

As the second kind of feature we use edge direction histogram. It is computed by grouping the edge pixels which fall into edge directions and counting the number of pixels in each direction. We are applying the Canny edge operator and consider 4 directional edges (horizontal, vertical and 2 diagonals) and 1 non-directional edge.

Since our subregions are arbitrary shaped we need to apply simple mirror padding to extend region to a rectangular area. Fig. 4 gives an example of extended subregions.

5.3 Texture features

Texture is another important property of images that helps in the image retrieval. We combine texture features with other visual attribute, because texture on its own does not have the capability of finding similar images. But it can classify textured images from non-textured ones.

In our work many subregions have same or very similar color, but they do not belong to the same semantic concept class. For example sky and water subregions have both similar shades of blue. Texture features help us classify subregion into the correct class. We applied one statistical and one transformed-based method.

5.3.1 Statistical method

We chose method based on co-occurrence matrices [2]. The co-occurrence matrix $C(i,j)$ shows the co-occurrence of gray-valued pixels i and j at a given distance d and given direction θ. In our case d is 1 and θ takes values $0°$, $45°$, $90°$ and $135°$. Together we have four different co-occurrence matrices from where six texture features are extracted: Energy, Contrast, Correlation, Difference Moment, Entropy and Homogeneity. They are defined as follows:

$$Energy = \sum_i \sum_j C(i,j)^2$$

$$Contrast = \sum_i \sum_j (i-j)^2 C(i,j)$$

$$Correlation = \frac{\sum_i \sum_j (ij) C(i,j) - \mu_i \mu_j}{\sigma_i \sigma_j}$$

$$Difference\ Moment = \sum_i \sum_j \frac{1}{1+(i-j)^2} C(i,j)$$

$$Entropy = -\sum_i \sum_j C(i,j) log C(i,j)$$

$$Homogeneity = \sum_i \sum_j \frac{C(i,j)}{1+|i-j|}$$

where

$$\mu_i = \sum_i i \sum_j C(i,j)$$

$$\mu_j = \sum_j j \sum_i C(i,j)$$

$$\sigma_i = \sum_i (i-\mu_i)^2 \sum_j C(i,j)$$

$$\sigma_j = \sum_i (j-\mu_j)^2 \sum_j C(i,j)$$

5.3.2 Transformed-based method

Another texture feature in our work is Gabor Texture Feature. The two-dimensional Gabor filter is defined as

$$Gab = \frac{1}{2\pi\sigma_x\sigma_y} e^{[-\frac{1}{2}((\frac{x}{\sigma_x})^2+(\frac{y}{\sigma_y})^2)+jW(x\cos\theta+y\sin\theta)]}$$

where σ_x and σ_y are scaling parameters of the filter, W is the radial frequency of the sinusoid and $\theta \in [0,\pi]$ specifies the orientation of the Gabor filters. Gabor filtered output $FGab$ of the image is obtained by the convolution of the given image F with Gabor function Gab for each of the orientation and scale. The magnitudes of the Gabor filters responses are represented by the mean and standard deviation:

$$\mu = \frac{1}{XY} \sum_{x=1}^{X} \sum_{y=1}^{Y} FGab$$

$$std = \sqrt{\sum_{x=1}^{X} \sum_{y=1}^{Y} \|FGab| - \mu|^2}$$

The feature vector is constructed using μ and std as feature components.

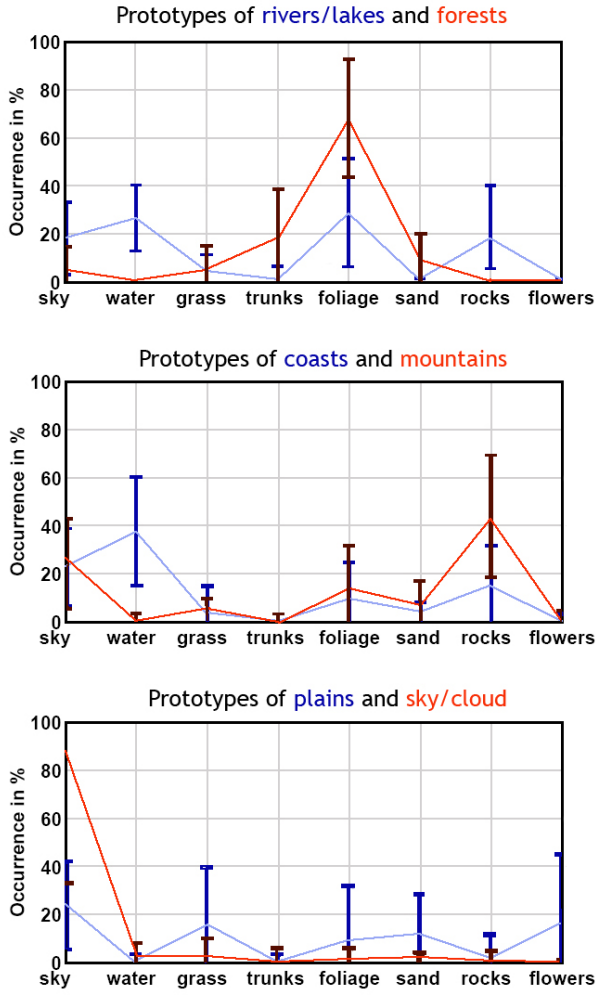


Figure 5: Prototypes of the six scene categories

5.4 Classification

We used two methods for the classification of the local semantic concepts, k-Nearest Neighbor and Support Vector Machine classifiers. Same classification methods were used in the initial method [12].

5.4.1 k-Nearest Neighbor classifier

The k-Nearest-Neighbor (kNN) classification is one of the most fundamental and simple non-parametric classification methods. For k-nearest neighbors, the predicted class of test sample x is set equal to the most frequent true class among k nearest training samples.

In our work we used the matlab implementation of kNN classifier. We tested several values of k. Best results were obtained by $k = 10$.

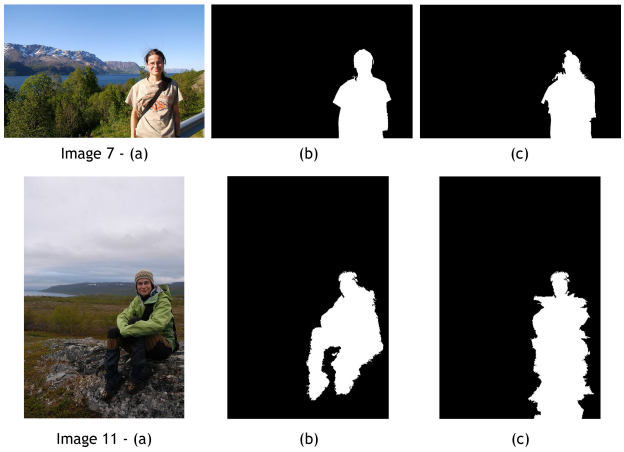


Figure 6: Human body detection (a) Original Image. (b) Manual detection. (c) Obtained result.

5.4.2 Support Vector Machine classifier

Support Vector Machines (SVM) are based on the concept of decision hyperplane. The SVM finds a linear separating hyperplane with a maximal margin in the higher dimensional space.

For our experiments, the LIBSVM package [1] with the radial basis function (RBF) kernel was employed. LIBSVM implements the "one-against-one" approach for multi-class classification. For $n = 8$ classes there are $\frac{n(n-1)}{2} = 28$ single classifiers and each one trains data from two classes. Each binary classification is considered to be a voting, where a new data point is allocated to the class with the highest number of votes.

6 Scene Categorization

The last stage of our method is scene categorization. Scene categorization refers to the task of grouping images or scenes into a set of given categories. In our work we have six different categories: coasts, forests, rivers/lakes, sky/clouds, plains and mountains (Figure 8 shows an example for each category). We define for each of these categories a category prototype. It is an example which is most typical for the respective category. Figure 5 displays these category prototypes and the standard deviations for each category.

Using the frequency of occurrence of eight semantic concept classes in the image the most similar category prototype is defined and that determines the high level scene category.

7 Results

This section summarizes the results of the proposed approach.

Overall **67,8%**	w	g	r	s	s	f	f	t
water	**71,8**	1,4	7,0	1,4	14,1	4,2	0,0	0,0
grass	9,2	**40,0**	7,5	0,8	0,0	18,3	23,3	0,8
rock	7,4	0,5	**77,5**	6,9	0,5	2,0	2,5	2,9
sand	0,0	6,7	23,3	**53,3**	10,0	3,3	0,0	3,3
sky	8,0	0,0	0,9	1,8	**82,1**	2,7	4,5	0,0
foliage	3,1	14,8	4,8	0,0	0,0	**71,6**	3,9	1,7
flowers	0,0	11,6	2,0	1,5	0,0	17,2	**67,7**	0,0
trunks	1,6	3,1	23,4	7,8	0,0	9,4	1,6	**53,1**
Precision	54,26	43,24	75,24	38,10	86,79	69,2	73,63	73,91

Table 1: Confusion matrix of the SVM concept classification (C=8, γ=0.125). Classification is in %

Color	52,3%
Co-ocurance matrix	41,2%
Gabor feature	43,4%
Edge direction	25,3%
Color+Co-ocurance matrix	59,8%
Color+Gabor feature	62,5%
Color+Edge direction	56,7%
All features	67,8%

Table 2: Low level feature relevance

We measured the quality of human body detection by comparing the obtained results with manual detection. We calculated the overlap and left-out feature. Overlap feature determines what percentage of the manual detection*(MD)* is covered by the obtained result*(OR)*.

$$Overlap = \frac{area(OR \cap MD)}{area(MD)}$$

Left-out feature determines what percentage of the obtained result is not covered by the manual detection.

$$Left-out = \frac{area(OR - MD)}{area(OR)}$$

Our method for human body detection was tested on 15 images and we achieved average Overlap $92,06\%$ and average Left-out $15,42\%$. The method works well if the person is standing straight. It is a typical pose on holiday pictures. If person is sitting or lying some errors may occur. (See Figure 6)

As a next step we tested which low level features are most relevant in classification process. Results obtained using SVM classifier can be find in Table 2. It is obvious that color feature give a good result, but its combination with texture feature leads in even better accuracy.

The ground truth for subregion membership to one of the eight semantic concepts was annotated manually. Together we annotated 1028 subregions. The class sizes vary from 54 (trunks) up to 192 (sky), because sky appears more often in the images than trunks. The classifiers are challenged with the inequality in the class sizes and the visual similarity of image regions that belong to different classes.

	Class size	Classification accuracy kNN	Classification accuracy SVM
sky	192	77,2%	82,1%
water	139	53,4%	71,8%
grass	111	20,7%	40,0%
trunks	54	43,8%	53,1%
foliage	166	66,7%	71,6%
sand	103	47,6%	53,3%
rocks	171	66,0%	77,5%
flowers	94	57,7%	67,7%

Table 3: kNN and SVM classification accuracies

The Table 3 shows that the SVM classification performs better than the kNN classification. We can see a correlation between the class size and the classification result. Sky, foliage, and rocks are the largest classes and they are also classified with the highest accuracy. In Table 1 is displayed confusion matrix of the SVM concept classification.

At the end we discuss results obtained by our proposed method and those obtained by the initial method [12]. Because of using a regular grid in the initial method, rectangular subregions belonging to two semantic concepts can be classified inaccurately. This is successfully improved by proposed method. On the other hand, in proposed method some problems occur in classification of small subregions.

For comparison, both methods were tested pixel-by-pixel with the manually annotated original image. An array of same size as original image was obtained, where logical 1 (white color) mean that pixels represented the same semantic category and logical 0 (black color) when different category. Initial method matched the ground truth in 68,05% compared to proposed method which reached 70,59%. An example can be find in Figure 7.

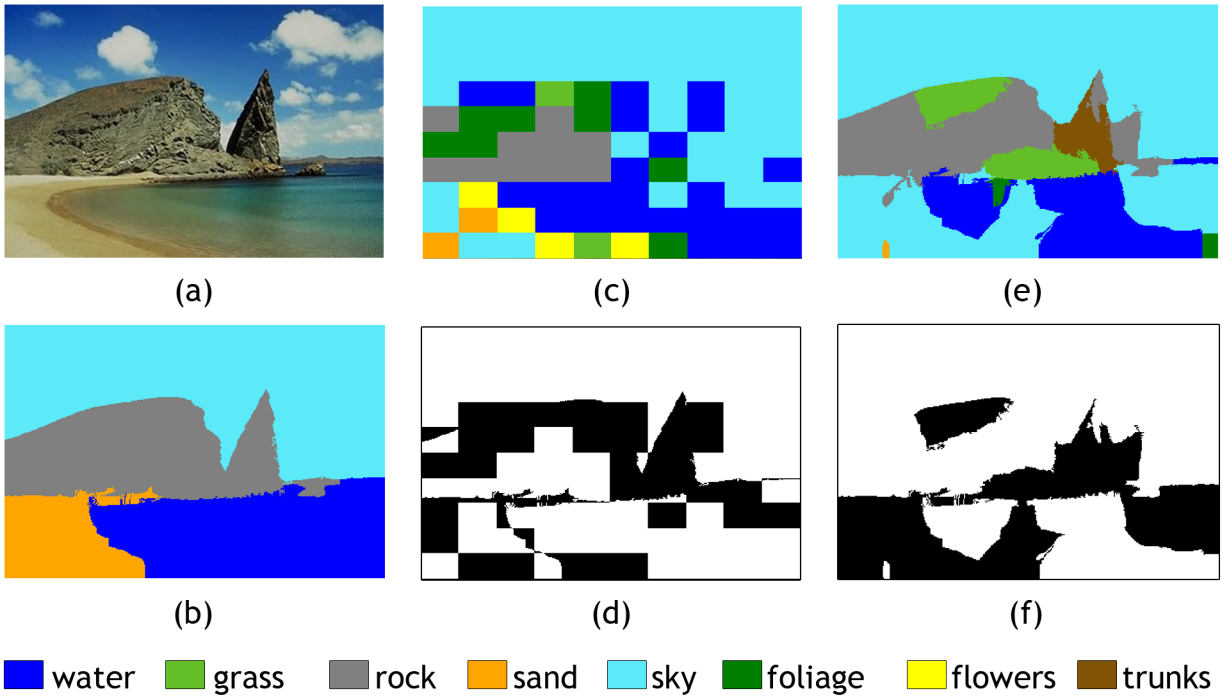


Figure 7: Local semantic concept classification (a) Original image. (b) Ground truth. (c+d) Result of initial method and equality map (e+f) Result of our proposed method and equality map

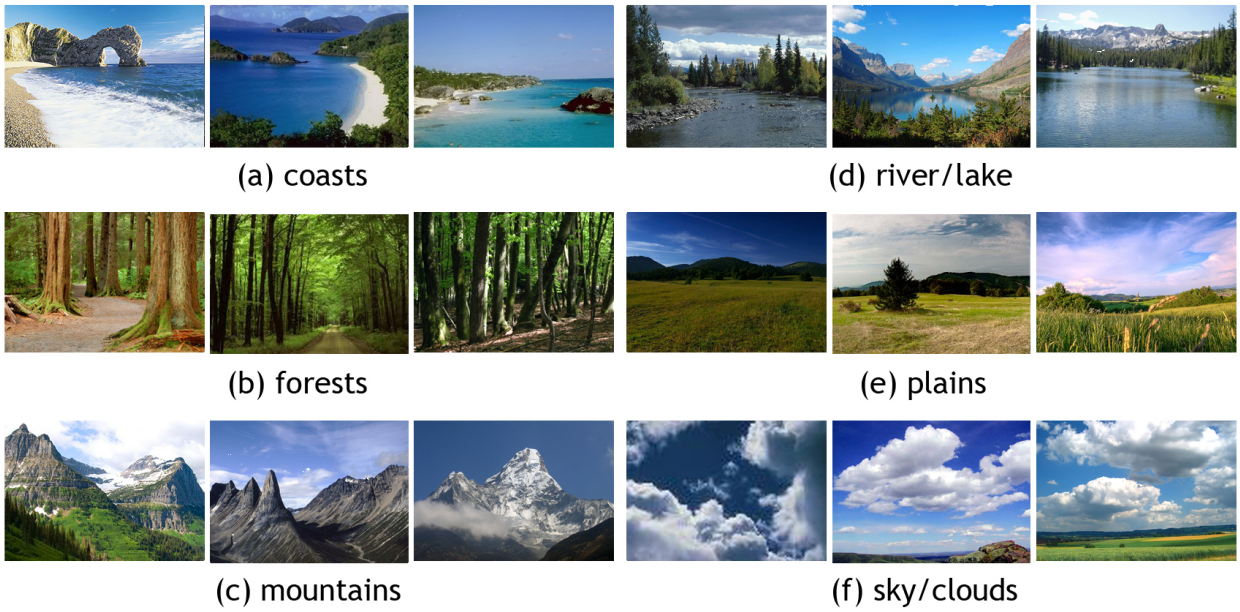


Figure 8: Exemplary images for each category

8 Conclusion

We implemented method for semantic categorization and retrieval of natural scene images presented in [12]. Since segmentation into 100 rectangular subregions used in this method can cause mistakes in semantic concepts classification, we modified this method by using segmentation into arbitrary shaped regions.

Our target images were typical holiday pictures from hiking outdoors. Due to frequent presence of family members in these pictures we enhance this method with automatic detection of people in the image.

9 Acknowledgment

The author wish to thank Elena Šikudová, PhD. for her support and the excellent leadership in this project.

References

[1] Ch. Chang and Ch. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] R. S. Choras. Image feature extraction techniques and their application for cbir and biometrics systems. *International Journal of Biology and Biomedical Engineering*, 1:6–16, 2007.

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

[4] V. Kazemi. Face detector (boosting haar features), 2010. `http://www.mathworks.com/matlabcentral/fileexchange/27150\-face-detector-boostinghaar-features`.

[5] H. B. Kekre and D. Mishra. Cbir using upper six fft sectors of color images for feature vector generation. *International Journal of Engineering and Technology*, 2:49–54, 2010.

[6] Y. Liu, D. Zhang, G. Lu, and W. Ma. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40(1):262 – 282, 2007.

[7] Jianjiang Lu, Zhenghui Xie, Ran Li, Yafei Zhang, and Jiabao Wang. A framework of cbir system based on relevance feedback. In *Proceedings of the 3rd international conference on Intelligent information technology application*, IITA'09, pages 175–178, Piscataway, NJ, USA, 2009. IEEE Press.

[8] V. Mezaris, I. Kompatsiaris, and M.G. Strintzis. An ontology approach to object-based image retrieval. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 2, pages II – 511–14 vol.3, sept. 2003.

[9] A. Mojsilovic and B. Gomes, J.and Rogowitz. Semantic-friendly indexing and quering of images based on the extraction of the objective semantic cues. *International Journal of Computer Vision*, 56:79–107, 2004.

[10] E. Šikudová. *On some possibilities of automatic image data classification*. PhD thesis, Comenius University, Bratislava, Slovakia, March 2006.

[11] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57:137–154, May 2004.

[12] J. Vogel and B. Schiele. Semantic modeling of natural scenes for content-based image retrieval. *Int. J. Comput. Vision*, 72:133–157, April 2007.

[13] J. Wu, Z. Lin, and M. Lu. Asymmetric semi-supervised boosting for svm active learning in cbir. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '10, pages 182–188, New York, NY, USA, 2010. ACM.

[14] Y. Zhuang, X. Liu, and Y. Pan. Apply semantic template to support content-based image retrieval. In *Proceeding of IST and SPIE Storage and Retrieval for Media Databases*, pages 23–28, 2000.

Image features in music style recognition

Kamil Behúň*
Supervised by: Michal Hradiš†

Department of Computer Graphics and Multimedia
Brno University of Technology
Brno / Czech Republic

Abstract

This work presents a novel approach to music style recognition inspired by feature extraction techniques used in image classification. To be able to utilize the image classification techniques, the 1D sound signal is transformed to its 2D representation — to a Mel-frequency spectrum. Small local areas of the spectrum are represented by 128-dimensional SIFT descriptors from which Bag of Words (BOW) representation of a whole signal is constructed. Dictionary used for translating the descriptors to the BOW representation is created by K-means algorithm. The BOW feature vectors are classified by non-linear Support Vector Machine classifier. The proposed approach was tested on publicly available music recognition data set GTZAN. The achieved results (86.4 % classification rate) compare favorably to previous results on this dataset — 84.3 % classification rate in (Panagakis et al., 2010), 74 % classification rate in (Holzapfel and Stylianou, 2008). Further, performance of the proposed audio parametrization was tested on video semantic category detection task from TRECVID evaluations, where it provides results superior to standard audio parametrizations.

Keywords: Music genre recognition, Mel-frequency spectrum, Local features, SIFT descriptors, Bag of Words, Support Vector Machine

1 Introduction

Automatic music style recognition has potential applications in on-line, as well as personal music databases. It can provide music genre information in cases where it is not available, and thus support navigation and searching in such music database. Furthermore, the problem of music genre recognition is related to the task of automatically suggesting suitable songs for users based on their particular taste or personalized song ratings.

A music genre or a music style is a conventional category of music works which all share some common attributes or origin (e.g. the period during which a musical composition was written, its instrumentation and treatment of those instruments, its form). Although the terms music genre and music style can be interpreted as being different, sometimes, they are used interchangeably. In the rest of the text, only the term genre is used, as the experiments are performed on a dataset of well established music genres (e.g. jazz, rock and country). Music genre recognition is a task, where the goal is to identify the attributes (features) specific for a particular genre, and estimate the genre of a given music piece based on these features.

Music genre recognition can be treated as a standard classification task which has two main parts - feature extraction and classification. Classification algorithms are usually task-independent and standard of-the-shelf classifiers can be used for most tasks. On the other hand, feature extraction is task-specific. It has to extract information from the classified data which is relevant for the particular task while suppressing effects non-informative sources of variation (e.g. lighting conditions in visual object detection, and compression or microphone quality in speaker recognition). For these reasons, feature extraction is usually hand-crafted for particular application, as is the case of music genre recognition [39]. However, several approaches have been recently suggested which learn features [28, 16]. These feature learning approaches were shown to be able to learn good representations for various data and achieve state-of-the-art results in wide range of classification tasks.

In this work, we explored different idea. We attempted to transfer well established features in one area to completely different task and data. Inspired by success of local image features [25] and Bag of Word representation [13] in image classification, we applied these methods to audio data. We show that the image features can be applied to audio data by transforming the 1D signals to spectrograms (2D representation), and that this type of feature extraction is able to achieve state-of-the-art results in music genre recognition, and that it surpasses standard audio features in semantic class detection as well.

Next section summarizes previous work in music genre recognition. Section 3.1 introduces the proposed feature extraction method and Section 3.2 discusses classification methods used in experiments. Section 4 presents datasets on which the approach has been evaluated and Section 5 describes the experiments and discusses the achieved re-

*xbehun03@stud.fit.vutbr.cz
†ihradis@fit.vutbr.com

sults. Finally Section 6 concludes the paper and outlines possible future work.

2 Previous work

When the music genre recognition is approached as a pattern recognition task, it consists of two parts. These parts are feature extraction and classification. This section describes previously proposed methods for classification and feature extraction for music genre recognition.

According to Tzanetakis at al. [39] features for music recognition can be divided into three basic types. These types are Pitch content features, Rhythmic content features and Timbral texture features. This classification is based on the type of information extracted by the features from audio signal.

Pitch content features characterize audio signals in terms of energy of different frequency bands. For example, in [39] the authors propose to detect few dominant pitches in a short time window [37] from which histograms of the dominant pitches across a whole song are computed. From the histograms, a small number of feature is extracted (e.g. amplitude of maximum peak, pitch interval between the two most prominent peaks).

Rhythmic content features represent rhythmic structure of the music. Method of Tzanetakis et al. [39] is based on detecting the most salient periodicities of the signal from which a beat histogram is created. Based on the beat histogram, several features are computed (e.g. relative amplitude of the two highest histogram peaks, frequency of the highest peaks). Another example of Rhythm content features is proposed in [20, 19, 22] where a 24-band psycho-acoustically modified spectrogram which reflects human loudness sensation is computed. Fourier transform is applied to each of the 24 frequencies to obtain spectra of loudness changes. Several types of features are then extracted from the loudness change spectra. The loudness change spectra is called by the authors Rhythm Pattern. The extracted features are average spectrum across the 24 frequency bands, statistical descriptors of each frequency band (mean, median, variance, skewness, kurtosis, min- and max- values), variance of each loudness change frequency across all 24 frequency bands, and temporal-variation of the Rhythm Patterns extracted from short (six second) time windows.

Timbral texture features should exhibit properties related to general timbre of the sound. They are based on short-time Fourier transform and they are calculated on short time frames of sound. This group includes many features, for example Spectral Centroid, Spectral Rolloff, Spectral Flux, Time Domain Zero Crossings [34], discrete wavelet transform coefficients [17, 23, 10], histogram of the wavelet coefficients [18], octave-based spectral contrast [40], and Mel-Frequency Cepstral Coefficients (MFCC). Mel-Frequency Cepstral Coefficients are very general features traditionally used in speech recognition [31, 29], and they provide good results in music genre recognition [19, 15, 8, 40, 39] as well.

Bergstra et al. [4], use combination of several Timbral texture features. After computing different frame-level Timbral texture features, they group non-overlapping blocks of consecutive frames into segments from which means and variances over a whole music signal are computed (means and variances are used as input to weak learners in AdaBoost).

The mentioned types features can be used individually; however, the best results are obtained by their combination. In [19], the authors combine features of different types (Pitch content features, Rhythmic content features, Timbral texture features) into hybrid features.

Many approaches which can not be clearly assigned to the above described groups exist. An example is the Bio-Inspired Joint Acoustic and Modulation Frequency Representation of Music [30]. In this method, an auditory temporal modulation representation is computed from an auditory spectrogram. This representation discards much of the spectro-temporal details, and focuses on the underlying slow temporal modulations of the audio signal.

Panagakis et al. [30] describe feature extraction methods called Multilinear Subspace Analysis Techniques. These techniques are Multilinear principal components analysis (MPCA), Non-negative tensor factorization (NTF), and Higher-order singular value decomposition (HOSVD) which are computed from tensors. Linear counterparts of these methods are Non-negative matrix factorization (NMF), Singular value decomposition (SVD) and Principal component analysis (PCA), which can be viewed as a special cases for first-order tensors (vectors).

Any type of classifier can be used for music genre recognition. In literature, the most commonly used classifier is Support Vector Machine (SVM) [20, 19, 22, 17, 18]. Compared to other classifiers, SVM was shown to give superior results [20, 17, 18]. Other classification methods used for music genre recognition are Gaussian mixture models [40, 39, 34], K-nearest neighbor classifier [20, 10, 39, 34] (K-NN), Round Robin ensemble [10] and Adaboost [4].

3 Method

As mentioned earlier, music genre recognition has two main parts — feature extraction and classification. The proposed feature extraction approach relies on transformation of 1D audio signal to 2D representation, and on consequent application of local feature extraction methods from the field of image classification. As a classifier, we selected SVM, because it provides state-of-the-art results in music genre recognition [20, 19]. Structure of our approach is shown in Figure 1, and a detailed description of each part of this structure is presented in the following text.

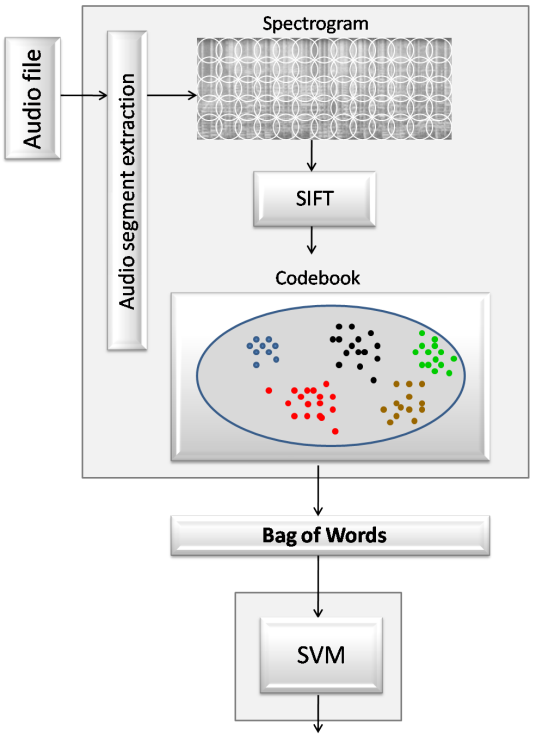


Figure 1: The processing pipeline: Audio signal is segmented, and Mel-frequency spectrograms are computed for the segments. Local features (SIFT) are extracted from the spectrograms on a regular grid. The local descriptors are translated to codewords by codebook, and Bag of Words representation is computed as a occurrence histogram of the codewords. The Bag of Words representation is used as an input to classification by SVM.

3.1 Feature extraction

This section presents a novel approach to music genre recognition inspired by feature extraction techniques used in image classification. Common approach in image classification [36] is to represent local parts of an image by a high-dimensional descriptors. Such descriptors encode appearance of the image patches, for example, spatial and directional distribution of gray-scale gradients as in SIFT descriptor [25, 24]. Such local descriptors are also called local image features in computer vision [33, 26]. To create a more compact representation, the local features can be assigned numerical identifiers based on their similarity to a set of prototypes [9]. The set of prototypes is called a visual codebook and the resulting image patches with assigned prototype IDs are called visual words.

Similarly to a text document, which can be described by the counts of individual words it contains, an image can be represented by counts of visual words [12]. Such representation is called Bag of Visual Words (BOW). BOW discards information about spatial relations. It discards any information about positions of the visual words, and retains only information of the local appearances.

To be able to use the local feature techniques from image classification, a 1D sound signal has to be transformed to its 2D representation. A natural way to create such 2D representation is to describe the sound signal in terms of energies of different frequency bands in short time windows. Methods for represent 1D signals in this way include short-term Fourier transform [2], wavelet transform [2] and many others. For the purpose of music genre recognition, we have chosen Mel-frequency spectrogram [38] as the 2D representation. The Mel-frequency spectrogram is similar to short-term Fourier transform, except the frequency scale is logarithmic which is closer to how humans perceive sounds (e.g. music intervals are multiplies of frequencies instead of fixed additions).

In our work, the Mel-frequency spectra were obtained by segmenting the audio track into 100 ms segments with 25 ms overlap. Mel-frequency spectra with 128 frequency bands and maximum frequency 12 KHz was computed from each of the segments. In order to be able to use existing local image feature methods, dynamic range and contrast of the spectrograms were reduced by

$$x = \left(\frac{\log(e+1)}{maxV}\right) * 255, \qquad (1)$$

where e is a value from the original Mel-frequency spectrum, $maxV$ is logarithm of the maximal value in the original Mel-frequency spectrogram and x is the resulting value in the lower dynamic range spectra. The transformation from equation 1 assures that the resulting values are in interval $< 0, 255 >$, and that they correspond to how humans perceive sound intensity (perception of acoustic intensity is logarithmic). Example of the obtained spectrogram is show in Figure 2.

Then the spectrograms were handled as images and local features were extracted from them. As the spectrograms do not exhibit any stable and distinct areas which could be detected by interest region detectors [27], we sampled the spectrograms on a regular grid with cell size 8×8 pixels and SIFT descriptors [24] were computed from the sampled small circular areas. The SIFT descriptor computes 16 Histograms of Oriented Gradients (HOG) on a 4 by 4 grid in the area of interest. Each of the histograms has 8 orientational bins, and magnitude of image gradient in each pixel is distributed between the closest bins according to its orientation and between the spatially closest histograms. The resulting SIFT descriptor is a vector of 128 values created by concatenating the 16 Histograms of Oriented Gradients.

To create a feature vector for a whole audio recording, extracted local features are aggregated to a Bag of Word (BOW) representation. In order to obtain the BOW representation, local features are first translated to visual words by codebook transform. To create a codebook, it is possible use to any clustering or quantization algorithm. The most commonly used is the k-means algorithm [9]. We

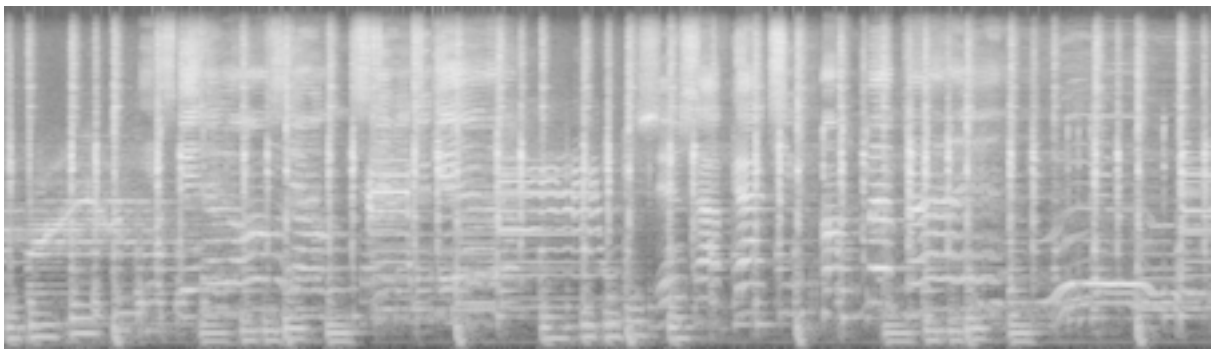


Figure 2: Example of a Mel-frequency spectrogram obtained for a music track. Columns are Mel-frequency spectra of signal segments in time. Lighter areas of the spectrogram represent higher energy. Bright vertical stripes represent the beats.

used k-means algorithm with Euclidean distance to obtain the set of prototypes which constitute the codebook - cluster centers become the prototypes. In the experiments, we used 4096 codewords (clusters) and initial positions of cluster centers were chosen randomly from training data set. Note that the goal of the k-means clustering in this case is not to find tight clusters of feature descriptors, rather, it provides good coverage of all possible descriptors, and it achieves low reconstruction error when the descriptors are quantized by the prototypes.

When assigning local features to the single closest codeword, quantization errors occur and some information is lost. This is especially significant in high-dimensional spaces, as is the case of the local patch descriptors where distances to several nearest codewords tend to be very similar. In the context of image classification, this issue was discussed for example by Gambert et al. [9], who propose to distribute the local descriptors to several close codewords according to codeword uncertainty. Computation of BOW with codeword uncertainty is defined for each codeword w from a codebook B as

$$UNC(w) = \sum_{p \in P} \frac{K(w,p)}{\sum_{v \in B} K(v,p)}, \tag{2}$$

where P is a set of local image features and K is a kernel function. We use Gaussian kernel

$$K(w,w') = exp\left(-\frac{\|w-w'\|_2^2}{2\sigma^2}\right), \tag{3}$$

where σ defines the size of the kernel. In our experiments, value of σ was set as the average distance between two closest neighboring codewords from a codebook. Equation 2 computes contributions of all local features to a particular word w and it sums the individual contributions. The BOW representation is a vector of $UNC(w)$ values for all words from a codebook normalized to unit length.

3.2 Classification

Our experiments were performed with Support Vector Machine classifier (SVM) [6] which is often used for various tasks in image and video classification [14, 33, 9, 36, 35]. SVM has four main advantages. It generalizes well, it can use kernels, it is easy to work with, and good-quality SVM solvers are available. Although non-linear kernels have been shown to perform better over the linear kernel in image recognition [32], we use the linear kernel in some experiments due to computational reasons. In addition to the linear kernel, Gaussian kernel

$$K(x,x') = exp\left(-\gamma\|x-x'\|_2^2\right), \tag{4}$$

was used in the experiments as well. Optimal value of the SVM regularization parameter C and the Gaussian kernel scale γ were estimated by grid search with 10-fold cross-validation with stratified sampling of training dataset.

Classifiers with the Gaussian kernel were trained using LIBSVM[1] [5] implementation of a solver for the standard soft-margin SVM formulation [6] which has a single regularization parameter C. Linear classifiers were trained using LIBLINEAR [7] which is able to handle large linear SVM problems efficiently.

4 Data sets

The approach, created for music genre recognition, was tested on GTZAN Genre collection[2]. This dataset was collected by G. Tzanetakis [39]. We chose GTZAN Genre collection, in order to be able to compare to previously published result on this dataset [30, 11, 3, 21, 4, 18, 39]. GTAZAN Genre collection contains 1000 audio tracks, where each track is 30 seconds long. These tracks are divided into 10 music genres — namely Blues, Classical,

[1] Experiments with Gaussan kernel SVM were concluded in RapidMiner which is a open–source tool for data minning, machine learning, text mining and other classification operations (http://rapid-i.com/).

[2] http://marsyas.info/download/data_sets

	Blues	Metal	Classical	Rock	Disco	Hiphop	Country	Jazz	Reaggae	Pop
Blues	**92**	0	1	2	0	2	2	5	2	2
Metal	0	**98**	1	0	0	2	0	0	0	0
Classical	2	0	**83**	3	0	3	0	6	2	10
Rock	4	1	3	**88**	3	1	1	6	3	2
Disco	0	0	0	1	**86**	0	1	3	3	1
Hiphop	0	1	2	0	0	**91**	0	0	1	2
Country	0	0	0	1	2	0	**94**	0	1	3
Jazz	0	0	0	1	3	0	0	**77**	3	3
Reggae	2	0	2	1	3	0	0	2	**83**	5
Pop	0	0	8	3	3	1	2	1	2	**72**

Table 1: Confusion matrix on the GTZAN genre collection achieved by parametrization for 32×32 local features (classification accuracy 86.4 %).

Country, Disco, HipHop, Jazz, Metal, Pop, Reggae, and Rock. Each genre is represented by 100 tracks. The tracks are all 22050 Hz mono 16-bit audio files.

Further, performance of the proposed audio parametrization was tested on data from semantic indexing task from TRECVID evaluations. The purpose of the semantic indexing task is to automatically detect semantic categories in video. Detection of semantic categories can be understood as video tagging which is, for example, performed by users when uploading videos to Internet archives (e.g. YouTube). The videos in semantic indexing task are tagged at the level of video shots.

In our experiments, we used a subset of TRECVID 2011 training data. First 20,000 shots from the TRECVID dataset were chosen as a training set, and following 50,000 shots were used for testing.

The TRECVID data was collaboratively annotated in an active learning setup[3] [1] which resulted in 346 usable semantic classes with enough annotations. From the annotated classes, we chose 55[4] the most common classes in our training set. The training set contains 29478 positive and 312398 negative class annotations. The testing set contains 86091 positive and 820057 negative annotations.

In the TRECVID data, some shots are very short while others are several minutes long. In order compensate for these difference, we chose a representative audio segment for a shot to be centered on the shot with duration restricted to be between 10 s and 20 s.

[3] http://mrim.imag.fr/tvca/

[4] The selected classes were: person, outdoor, face, vegetation, single person, male person, adult, indoor, trees, daytime outdoor, overlaid text, female person, computer or television screens, plant, entertainment, sky, building, vehicle, suburban, singing, ground vehicles, landscape, streets, road, celebrity entertainment, actor, walking running, instrumental musician, scene text, sitting down, animal, child, waterscape waterfront, car, doorway, news studio, walking, politics, cityscape, two people, hand, city, quadruped, mammal, carnivore, explosion fire, reporters, mountain, domesticated animal, female-human-face-closeup, dark-skinned people, graphic, crowd, chair, teenagers

5 Experiments and results

This section contains description of experiments and results in genre recognition and semantic indexing. Our main experiment for both tasks was aimed to explore the effect of local feature size, in order to estimate what size of the local features is the best for the solved tasks (small size is capture detail and bigger sizes represent larger context). Results achieved on the GTZAN dataset were compared to results of other music genre recognition approaches, and result achieved on TRECVID 2011 dataet were compared to two standard audio parametrizations.

Following the experimental setup used in [30, 39], stratified tenfold cross validation is employed to estimate performance in experiments conducted on the GTZAN dataset. That means, each training set contained 900 tracks (9 parts of validation), testing set contained 100 tracks (1 part of validation) from GTAZAN, and the parts were gradually alternated for training and testing in the experiment. This experimental setup allows for comparison to other published approaches.

We did experiments for 8×8, 16×16 and 32×32 pixels sizes of the local features extracted from spectograms. SVM with RBF kernel function was used. Results for these different sizes are shown in Table 4. The best classification accuracy (86.4 %) was achieved by parametrization for 32×32 local features. Confusion matrix for this result is shown in Table 1.

Size of loc. features	Classification accuracy
8×8	83.4 %
16×16	84.2 %
32×32	**86.4 %**

Table 4: Classification accuracy achieved on GTZAN Genre collection for different sizes of local features.

Table 2 compares classification accuracy on the GTZAN Genre collection to other approaches. As can be seen, our approach performs the best. The results show that the local image features are a good choice for music

Approach (features + classifier)	Classification accuracy
mel-spectrogram - SIFT+ SVM (this work)	**86.4**%
non-negative MPCA + SVM [30]	84.3%
aggregate features + AdaBoost [4]	82.5%
wavelet histograms + SVM [18]	78.5%
audio and symbolic features + SMV [21]	76.8%
many features + NTF [3]	75.0%
spectrogram - NMF + GMM [11]	74.0%
wavelet histograms + LDA [18]	71.3%
pitch, rhytmic and timbral features + GMM [39]	61.0%

Table 2: Classification accuracies achieved by our approach and other published approaches for GTZAN Genre collection.

Features	Mean Average Precision
mel-spectrogram - SIFT 8×8 + SVM	0.144
mel-spectrogram - SIFT 16×16 + SVM	**0.147**
SPEC + SVM	0.115
MFCC + SVM	0.113

Table 3: Mean Average Precisions achieved in semantic indexing task on TRECVID 2011 training set.

genre recognition.

To compare our approach with a standard baseline audio parametrization on TRECVID 2011, local features with sizes of 8×8 and 16×16 were used. SVM with linear kernel function was used as a classifier for BOW, as well as, for the standard audio parametrizations. Meta parameter was estimated by stratified sixfold cross validation.

The two baseline parametrizations are SPEC and MFCC which are both computed from short overlapping time windows. A spectrum is computed for each window a for SPEC, respectively mel-frequency spectrum for MFCC. Bag of Words representation is computed for the obtained spectra — one spectra is translated one code word. Both parametrizations use 100 ms window with 75 ms overlap, and a codebook of 4096 words. Performance of the SPEC, MFCC and our approach are compared in Table 3. The table shows mean average precision for each approach. Mean average precision (MAP) is the mean value of average precision scores across all classes. The average precision is an area under precision-recall curve. As can be seen in Table 3, the best MAP 0.147 was achieved by the local image features with size 16×16 pixels.

6 Conclusions

This paper presented a novel feature extraction approach for audio data which is based on SIFT local features and Bag of Word representation which were previously used with good results in image classification. The experiments show that the proposed feature extraction provides very good results in music genre recognition task. Specifically, its performance is superior to previously published results on the GTZAN dataset. In semantic indexing, the proposed approach provides better results compared to MFCC and spectral features. Overall, the results are very promising and we plan to extend the work to other classification tasks which process audio data. Moreover, image classification provides wide variety of existing features which could all be applied to spectrograms as well, and which could possibly surpass the presented approach. Previous work in image classification suggests that no particular type of features is optimal in wide variety of tasks, and that combining different types of features improves results [36] (at the expense of higher computational cost). It is probable that combining multiple features inspired by image classification in music recognition would improve results even further. Furthermore, the features could be combined with traditional audio features.

References

[1] S. Ayache and G. Quénot. Evaluation of active learning strategies for video indexing. *Image Communication*, pages 692–704, 2007.

[2] T. Bartosch and D. Seidl. Spectrogram analysis of selected tremor signals using short-time Fourier transform and continuous wavelet transform. *Annali Geofisica*, pages 497–506, 1999.

[3] E. Benetos and C. Kotropoulos. A tensor-based approach for automatic music genre classification. *European Conference on Signal Processing 2008*, 2008.

[4] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kégl. Aggregate features and ADABOOST for music classification. *Machine Learning*, pages 473–484, 2006.

[5] Ch. Chang and Ch. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, page 27, 2011.

[6] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, pages 273–297, 1995.

[7] R. Fan, K. Chang, Ch. Hsieh, X. Wang, and Ch. Lin. LIBLINEAR: A Library for Large Linear Classification. *The Journal of Machine Learning Research*, pages 1871–1874, 2008.

[8] J. Garcia, A. Barbedo, and A. Lopes. Automatic genre classification of musical signals. *EURASIP J. Appl. Signal Process.*, page 157, 2007.

[9] J. Gemert, C. Veenman, A. Smeulders, and J. Geusebroek. Visual Word Ambiguity. *PAMI*, pages 1271–1283, 2010.

[10] M. Grimaldi, P. Cunningham, and A. Kokaram. A Wavelet Packet representation of audio signals for music genre classification using different ensemble and feature selection techniques. In *Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 102–108, 2003.

[11] A. Holzapfel and Y. Stylianou. Musical Genre Classification Using Nonnegative Matrix Factorization-Based Features. *IEEE Transactions On Audio Speech And Language Processing*, pages 424–434, 2008.

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, pages 2169–2178, 2006.

[13] N. Lazic and P. Aarabi. *Importance of Feature Locations in Bag-of-Words Image Classification*, pages I–641 – I–644. 2007.

[14] Q. Le, W. Zou, S. Yeung, and A. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1–4, 2011.

[15] Ch. Lee, J. Shih, K. Yu, and H. Lin. Automatic music genre classification based on modulation spectral analysis of spectral and cepstral features. *IEEE Transactions on Multimedia*, pages 670–682, 2009.

[16] H. Lee, P. Pham, and A. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. *Neural Information Processing Systems (NIPS)*, pages 1–9, 2009.

[17] T. Li and M. Ogihara. Toward intelligent music information retrieval. *IEEE Transactions on Multimedia*, pages 564–574, 2006.

[18] T. Li, M. Ogihara, and Q. Li. A comparative study on content-based music genre classification. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 282–289, 2003.

[19] T. Lidy, C. Silla Jr., O. Cornelis, F. Gouyon, A. Rauber, C. Kaestner, and A. Koerich. On the suitability of state-of-the-art music information retrieval methods for analyzing, categorizing and accessing non-Western and ethnic music collections. *Signal Processing*, pages 1032–1048, 2010.

[20] T. Lidy, R. Mayer, A. Rauber, P. León, A. Pertusa, and J. Quereda. A Cartesian Ensemble of Feature Subspace Classifiers for Music Categorization. In *ISMIR*, pages 279–284. International Society for Music Information Retrieval, 2010.

[21] T. Lidy, A. Rauber, A. Pertusa, and J. Inesta. MIREX 2007: Combining Audio And Symbolic Descriptors For Music Classification From Audio. In *MIREX 2007 - Music Information Retrieval Evaluation eXchange, MIREX Genre Classification Contest.*, Vienna, Austria, 2007.

[22] T. Lidy, A. Rauber, A. Pertusa, P. León, and J. Inesta. Audio Music Classification Using A Combination Of Spectral, Timbral, Rhythmic, Temporal And Symbolic Features. *Artificial Intelligence*, pages 5–7, 2008.

[23] Ch. Lin, S. Chen, T. Truong, and Y. Chang. Audio Classification and Categorization Based on Wavelets and Support Vector Machine. *IEEE Transactions on Audio, Speech Language Processing*, pages 644–651, 2005.

[24] D. Lowe. Object Recognition from Local Scale-Invariant Features. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1150, 1999.

[25] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, pages 91–110, 2004.

[26] K. Mikolajczyk and C. Schmid. A Performance Evaluation of Local Descriptors. *IEEE Pattern Analysis and Machine Intelligence*, pages 1615–1630, 2005.

[27] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Gool. A Comparison of Affine Region Detectors. *Int. J. Comput. Vision*, pages 43–72, 2005.

[28] A. Mohamed, T. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny. Deep Belief Networks using discriminative features for phone recognition. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5060–5063. IEEE, 2011.

[29] L. Muda, M. Begam, and I. Elamvazuthi. Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques. *Computing Research Repository (CoRR)*, pages 138–143, 2010.

[30] Y. Panagakis, C. Kotropoulos, and G. Arce. Non-Negative Multilinear Principal Component Analysis of Auditory Temporal Modulations for Music Genre Classification. *IEEE Transactions On Audio Speech And Language Processing*, pages 576–588, 2010.

[31] I. Patel and Y. Rao. Speech Recognition Using Hidden Markov Model with MFCC-Subband Technique. pages 168–172, 2010.

[32] F. Perronnin, J. Senchez, and Y. Xerox. Large-scale image categorization with explicit data embedding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2297–2304, San Francisco, CA, 2010.

[33] K. Sande, T. Gevers, and C. Snoek. Evaluating color descriptors for object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1582–1596, 2010.

[34] E. Scheirer and M. Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, pages 1331–1334, 1997.

[35] A. Smeaton, P. Over, and W. Kraaij. High-Level Feature Detection from Video in TRECVid: a 5-Year Retrospective of Achievements. In *Multimedia Content Analysis, Theory and Applications*, pages 151–174. Springer Verlag, Berlin, 2009.

[36] C. Snoek, K. Sande, O. Rooij, B. Huurnink, E. Gavves, D. Odijk, M. Rijke, T. Gevers, M. Worring, D. Koelma, and A. Smeulders. The MediaMill TRECVID 2010 Semantic Video Search Engine. In *TRECVID 2010: Participant Notebook Papers and Slides*, 2010.

[37] T. Tolonen and M. Karjalainen. A computationally efficient multipitch analysis model. *IEEE Transactions On Speech And Audio Processing*, pages 708–716, 2000.

[38] V. Tyagi, I. McCowan, H. Misra, and H. Bourlard. Mel-cepstrum modulation spectrum (MCMS) features for robust ASR. In *Automatic Speech Recognition and Understanding, 2003. ASRU '03. 2003 IEEE Workshop on*, pages 399–404, 2003.

[39] G. Tzanetakis and P. Cook. Musical Genre Classification of Audio Signals. *Speech and Audio Processing, IEEE Transactions on*, pages 293–302, 2002.

[40] K. West and S. Cox. Features and classifiers for the automatic classification of musical audio signals. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, 2004.

3D Data Processing

Non-Linear Dimensionality Reduction With Isomaps

Galina Paskaleva*
Supervised by: Stefan Jeschke

Institute of Computer Graphics
Vienna University of Technology
Vienna / Austria

Abstract

This paper discusses the Isomap method for dimensionality reduction and studies its performance on both artificial and natural datasets. While linear methods for dimensionality reduction such as Principle Component Analysis (PCA) detect a linear subspace of the original domain that represents the data with maximal accuracy, the Isomap method detects the tangent space of a manifold embedded in the original domain. PCA remains globally linear as it simply transforms the data from one vector space into another. Isomap is only locally linear; globally it maps a low-dimensional manifold embedded in a high-dimensional domain into a lower-dimensional vector space by globally aligning the local manifold tangent spaces. The critical precondition for its success is that the sampling frequency of the data is sufficient to avoid 'short-circuits' in the spatial dissimilarity matrix. This is demonstrated by means of artificial datasets with dimensionality ranging from two (e.g. planar geometry) to several thousand (e.g. images of geometrical models) since they allow absolute control over the sampling frequency. The results of Isomap applied to a natural dataset (pressure images from a foot scan) are much more ambiguous. Since we have no prior knowledge of the 'hidden' dimensionality of the data we do not know if the sampling frequency is sufficient. The difficulty here lies in differentiating between classification outliers, i.e. atypical samples, and 'unexpected' degrees of freedom, that may initially seem 'wrong' if they contradict our expectations.

Keywords: Isomap, Principle Component Analysis, dimensionality reduction, non-linearity, sampling frequency

1 Introduction

Dimensionality reduction can be viewed as the process of finding the domain best suited for embedding a set of high-dimensional data samples [2]. Ideally, the dimensionality of this domain is considerably lower (e.g. an order of magnitude or more) than the dimensionality of the data samples, yet sufficient to reproduce them within a given margin of error. A highly useful byproduct of dimensionality reduction is an interpretation of the given data set within the new domain [2] that is not possible in the original domain due to its high complexity. The purpose of this work is to discuss the Isomap method for dimensionality reduction and to compare it to the classical method of Principal Component Analysis (PCA) in terms of effectiveness with particular emphasis on the quality of insight provided into both geometrical and image data.

Human visual perception is capable of reducing complex data to its underlying dimensionality in less than a second (i. e. the lighting in a photograph), particularly in the presence of prior knowledge or expectations [9, 3]. While during the first phases of object recognition linear methods such as frequency decomposition play an important role [9], in the latter stages of conscious search for yet undiscovered interdependencies they might prove too restrictive. In this work the ability of the most commonly used linear method (PCA) and one non-linear method (Isomaps) to help a human observer gain new insight into complex high-dimensional (visual) data is examined in detail.

The next section provides a brief overview over the existing methods for dimensionality reduction. Section 3 reviews Isomaps and PCA in detail. Section 4 compares the results of the application of both Isomaps and PCA to artificially produced data sets. Section 5 performs the same comparison on the basis of a natural data set. Section 6 discusses the results.

2 Methods for Dimensionality Reduction

All methods for dimensionality reduction can be divided into two large classes - the classical linear techniques and the more recently developed non-linear techniques.

As the term *linear* will be used extensively we provide a definition. A linear map ($A : X \to Y$ with X and Y vector spaces) or a linear operator ($A : X \to X$ on the vector space X) is a function with the following two properties:

additivity :	$A(\vec{x}+\vec{y}) = A(\vec{x}) + A(\vec{y})$
homogeneity of degree 1:	$A(\vec{x}\alpha) = (A\vec{x})\alpha$ with $\alpha \in R$

Whenever a reduction step is described as linear it is to

*g.paskaleva@gmx.at

be understood that it defines a linear map from the vector space of the original data points X into a new vector space Y.

2.1 Linear Methods for Dimensionality Reduction

The classical methods for dimensionality reduction include PCA, classical Multidimensional Scaling (MDS), Factor Analysis (FA), and Independent Component Analysis (ICA). What is common to all of them is that they look for a linear subspace in the sampled data [2, 5]. A common advantage of all linear methods is that the embedding of new data samples in the already calculated subspace is reduced to applying the linear mapping to the new samples. For most non-linear methods, however, only estimation techniques exist [5].

2.2 Non-Linear Methods for Dimensionality Reduction

Methods that can deal with non-linear cases can be subdivided in techniques that preserve global data properties in the low-dimensional representation - e.g. Kernel PCA, Semidefinite Embedding (SDE), MDS with a non-Euclidean distance function, Isomaps, and Maximum Variance Unfolding (MVU), techniques that preserve local properties in the low-dimensional representation - e.g. Locally Linear Embedding (LLE), Laplacian Eigenmaps (LEM), Hessian LLE, and techniques that calculate local linear models and subsequently align them globally. Markov processes and neural networks have also been successfully used for non-linear dimensionality reduction. Detailed discussion of the above methods can be found in [2, 5, 8].

3 Mathematical Background of PCA and Isomaps

In this section the mathematical theory behind PCA and Isomaps is discussed in detail. The fundamental difference between the two methods is presented in Figure 1.

3.1 PCA

Let us assume that there are N data samples of dimensionality D and let the i-th data point be represented as a D-dimensional vector $\vec{x}_i = (x_{i1}, x_{i2}, ..., x_{iD})$ with $i = 1, ..., N$. The result of PCA is a linear map that projects the data points from the original vector space into another while retaining as much of the variability of the data as possible and orienting the coordinate axes (or principal components) spanning the new vector space according to it. The principal components are linear combinations of the existing data points. As they are orthogonal to each other there can be at most D of them.

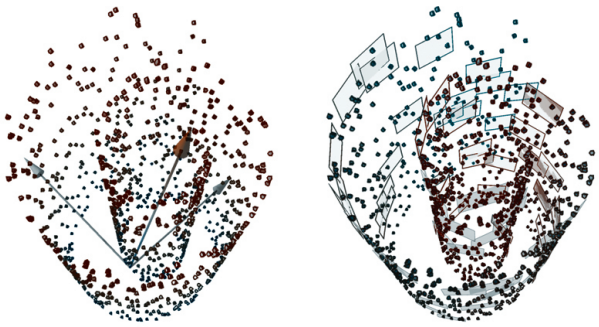


Figure 1: On the left PCA determines an optimal embedding of the 'Swiss roll' dataset in a 3-dimensional vector space. On the right Isomap performs local linear fitting before mapping the detected local vector spaces into a global vector space.

Let us assume that the axis aligned to the maximum variability of the data is $P_1 = \sum_{i=1}^{n} (w_i \vec{x}_i)$.

If we combine the coefficients w_i in a vector $\vec{w} = (w_1, w_2, ..., w_N)$ and the data points $\vec{x}_i$ - column-wise in a $N \times D$ matrix X we can write the previous equation as $P_1 = w^T \times X$.

The variance along P_1 can be calculated as $VarP_1 = E[(P_1 - E[P_1])^2]$ where $E[P_1]$ is the expected value (mean) of P_1, which exists whenever P_1 has a variance [7]. We can assume $E[P_1] = 0$ since this would only remove one translation from the linear map that can be re-applied in the very last step. Thus

$$\begin{aligned} VarP_1 &= E[P_1^2] \text{ or} \\ &= (\vec{w}^T \times X) \times (\vec{w}^T \times X)^T \\ &= \vec{w}^T \times (X \times X^T) \times \vec{w} \\ &= \vec{w}^T \times S \times \vec{w} \end{aligned}$$

with S the (scaled) covariance matrix of the original data points [7].

As $VarP_1$ is not bounded above for an arbitrary $\vec{w}$ we impose the restriction $\|\vec{w}\| = 1$. We want to derive P_1 from the covariance of the data samples, not scale it to infinity. In order to maximize $\vec{w}^T \times S \times \vec{w}$ with the constraint $\vec{w}^T \times \vec{w} = 1$ we apply a Lagrange multiplier α_1:

$$L(\vec{w}, \alpha_1) = \vec{w}^T \times S \times \vec{w} - \alpha_1 \times (\vec{w}^T \times \vec{w} - 1).$$

After differentiating with respect to $\vec{w}$ we have $S \times \vec{w} = \alpha_1 \times \vec{w}$. Obviously, $VarP_1$ is maximal for the biggest α_1, i.e. the largest eigenvalue of S corresponding to the eigenvector $\vec{w}$.

Essentially, in its first step PCA calculates the $D \times D$ covariance matrix of the original data samples. Then the R eigenvectors ($R \leq D$) corresponding to the R largest non-zero eigenvalues of the covariance matrix build the base B_R of the R-dimensional linear domain in which the data samples can be projected by means of the map $B_R B_R^T$ with minimal loss of variability [2]. A significant difference between the Rth and the $(R+1)$th largest eigenvalue indicates the existence of an R-dimensional linear subspace 'hidden' in the original domain. In that case the remaining

$D-R$ Eigenvectors can be interpreted as noise obscuring the true dimensionality of the data.

An obvious drawback of PCA is that the size of the covariance matrix depends on the dimensionality of the data samples and becomes impractical for high-dimensional domains such as image data. One way to avoid this, especially if $N < D$, is to apply dual PCA [5].

The projection calculated by PCA has the effect of minimizing the squared reconstruction error. Using this property as a starting point, Ghodsi shows in [2] that the linear map of PCA can be obtained as the matrix U in the singular value decomposition of $X = U \times E \times V^T$ as it contains the eigenvectors of XX^T (V contains the eigenvectors of X^TX and E - the square roots of the non-zero eigenvalues of both XX^T and X^TX in its diagonal). Once U, V and E can be reduced in size according to the rank of E it follows that E is invertible and $U = XVE^{-1}$. This produces the linear map U^TU of dual PCA (by solving the dual problem to that of classical PCA).

3.2 Isomaps

In its first step Isomap uses a neighborhood criterion (k nearest neighbors or an ε-neighborhood [4]) to build a neighborhood graph of the data samples. Then it builds a $N \times N$ spatial dissimilarity matrix by calculating the shortest paths between each pair of data samples, which allows non-linear subspaces to come into consideration [6]. The goal is to use the shortest paths between data samples to detect a manifold embedded in the vector space of the original data points.

In its second step Isomap applies classical MDS. This method takes the $N \times N$ dissimilarity matrix of N D-dimensional data points produced in the previous step as its input and calculates a linear transformation that preserves the pair-wise shortest path distances between the points while disregarding their coordinates in the original domain entirely. In effect, instead of minimizing the reconstruction error relative to the the original coordinates (which is what PCA does) MDS minimizes the reconstruction error relative to the dissimilarity matrix. Since it contains only scalar values it does not depend on the original dimensionality of the data points; in fact, it is the function of MDS to find the minimal suitable dimensionality for data points with (nearly) the same dissimilarity matrix (see Figure 2).

MDS uses the fact that the dissimilarity matrix, when employing the Euclidean distance as a dissimilarity measure, is expressed by means of the inner product of the $N \times D$ data point matrix X (the Gram matrix [8]): $G = X^TX = -1/2 \times H \times D^2 \times H$ with $H = I - 1/D \times \vec{e} \times \vec{e}^T$ where $\vec{e} = (1,1,...,1)$ is a N - dimensional vector [5]. There exist multiple methods for recovering a data matrix X when only the Gram matrix G is known [5, 8]. The result corresponds to a 'flattening' or linearization of the tangent space of any manifold concealed in the original domain.

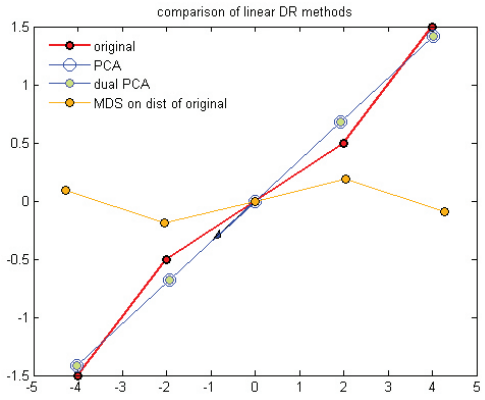


Figure 2: A comparison of PCA, dual PCA and MDS. Both PCA methods perform reconstruction with one component. MDS detects two degrees of freedom.

Tennenbaum *et. al.* provide a proof in [1] that as the number of data samples increases so does also the accuracy of the approximation of the underlying manifold performed by Isomap. Asymptotic convergence to the 'true' underlying structure is guaranteed for smooth manifolds isometric to a convex domain of Euclidean space [4]. These dependencies will be demonstrated in the next section.

4 PCA and Isomap Applied to Artificial Datasets

The experiments on artificial datasets documented in [5] show that Isomap is significantly better than PCA in detecting non-linear structures on the 'Swiss roll' , the 'Helix' (i.e. on smooth highly non-linear manifolds, the first one Euclidean, the second one - non-Euclidean) as well as on a discontinuous 'broken Swiss roll' dataset. However, Isomap performs only slightly better than PCA on a dataset with high intrinsic dimensionality.

4.1 Low-dimensional Data Sets

We start our experiments with a one-dimensional dataset folded in two dimensions - a spiral. Applying Isomap with 4 nearest neighbors produces the mapping (represented by the straight lines) shown in Figure 3.

The next experiment applies Isomap with 4 nearest neighbors again to a one-dimensional set, but this time it

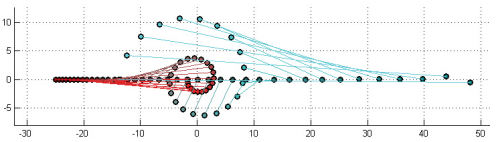


Figure 3: Isomap performed on a one-dimensional manifold folded in two dimensions.

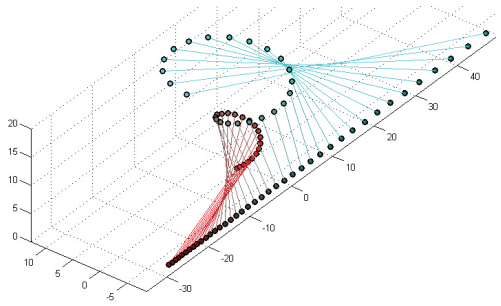


Figure 4: Isomap performed on a one-dimensional manifold folded in three dimensions.

is folded in three dimensions. As Figure 4 shows the result is the same as in the previous case.

Figure 5 shows a region in the data samples that causes the error in the mapping of the manifold into the one-dimensional domain (the loop in Figure 6 - the color gradient corresponds to the ordering of the data samples according to the first detected degree of freedom). Since the underlying manifold is actually one-dimensional, the loop represents the failure of Isomap to recognize this in the vicinity of the 'short-circuit'.

These three examples demonstrate a key requirement for the convergence of Isomaps [1]: *the sampling condition*. It states that arbitrarily close approximation of the underlying manifold M is possible if the neighborhood graph contains all edges connecting data samples at a distance less than a positive parameter ε, and for each point m on M there exists a data sample that is at a distance less than $\varepsilon/4$ from m. In other words, the sampling density must be sufficient for every point on the manifold. In particular, it must be at most half of the shortest distance between manifold 'folds' (see Figure 5).

The next two experiments utilize the 'Swiss roll' dataset

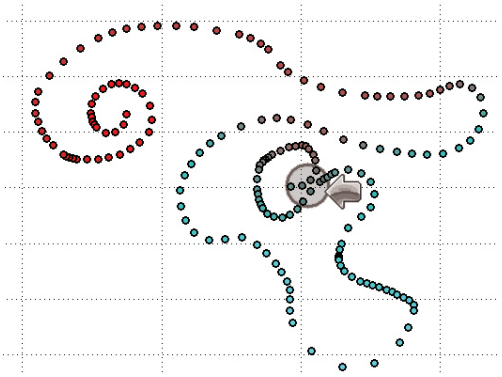


Figure 5: An example of insufficient sampling density.

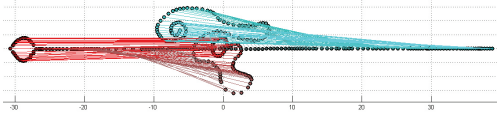


Figure 6: The loop on the far left showcases the error resulting from insufficient sampling density.

used in [4] and [5]. Figures 7 and 9 visualize the difference resulting from the choice of the number of nearest neighbors k. For $k = 7$ Isomap manages to recognize the underlying two-dimensionality of the dataset in spite of the numerous 'short-circuits' resulting from insufficient data sampling (see Figure 7 - the change in color corresponds to the ordering along one detected degree of freedom, the edges are those of the neighborhood graph). The 'unwrapped' manifold is shown in Figure 8. For $k = 20$, however, we obtain the same result as with PCA - there is no meaningful dimensionality reduction, only a change in the coordinate system. Since for all examples above PCA cannot reduce the dimensionality of the data (as there is no significant difference in the detected eigenvalues) its results are not shown.

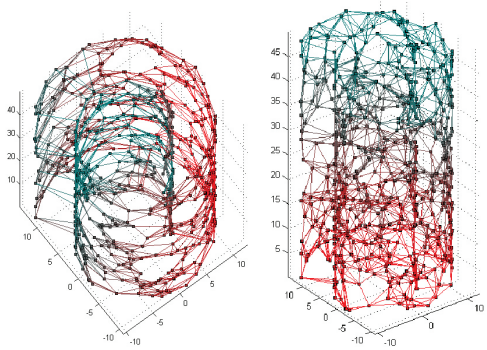


Figure 7: The 'Swiss roll' dataset colored according to the first and second degrees of freedom detected by Isomaps with $k = 7$.

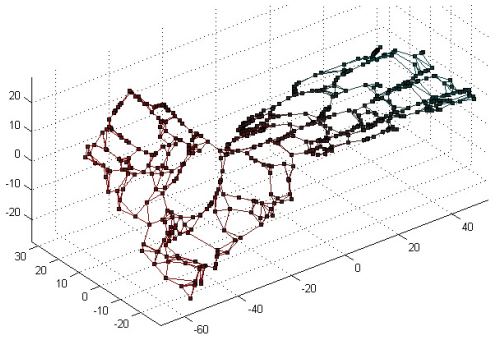


Figure 8: The unwrapped 'Swiss roll' dataset from Figure 7 shows a slight 'thickening' in the third dimension due to 'short-circuiting' errors.

The 'Swiss roll' examples showcase another important condition for successful approximation: *the neighborhood condition*. It requires that the neighborhood graph does not contain edges connecting data samples whose Euclidean distance is larger than certain parameters derived from the minimum radius of curvature r_0 and the minimum branch separation s_0 of M. The minimum branch separation is the largest Euclidean distance between points on M that still guarantees a geodesic distance of less than πr_0 [1]. In the case of 20 nearest neighbors this condition is clearly violated and thus Isomap fails.

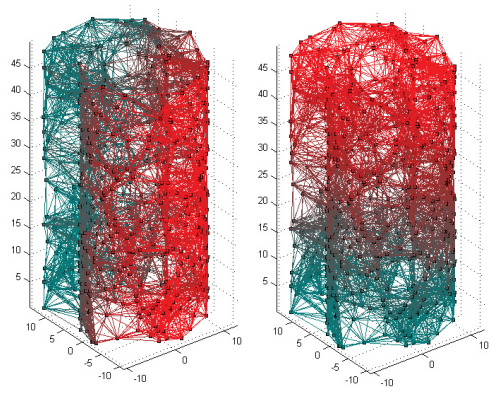


Figure 9: The 'Swiss roll' dataset colored according to the first and second degrees of freedom detected by Isomaps with $k = 20$.

In [1] Bernstein *et. al.* provide a proof that a random sampling of sufficient density can satisfy the two conditions above with probability depending on the volume of M and on the volume of the smallest metric ball in M. However, while these conditions are easy to check in an artificially generated data set with known degrees of freedom it is quite difficult to do so for natural datasets where the prior knowledge is insufficient, as will be demonstrated in Section 5.

4.2 High-dimensional Data Sets

We now move to a very high-dimensional domain - that of image data. The generated images satisfy both the sampling and the neighborhood conditions from Section 4.1.

We generate 300 images with resolution 64×86 pixels of a glass ball illuminated by several light sources (as shown in Figure 11), only one of which changes position - it traverses one half of a circular trajectory from left to right. The resulting data samples are 5504-dimensional vectors ($64 \times 86 = 5504$) containing 8-bit integral intensity values.

Figure 10 shows the residual variances after consecutive data reconstruction with 1 to 6 components detected by PCA and by Isomap ($k = 2$ nearest neighbors considered in the neighborhood graph) respectively. Where PCA performs a frequency decomposition of the data (first image group) and the residuals diminish rather gradually, Isomap (second image group) shows a sharper drop in the residuals after the first detected degree of freedom and correctly identifies the main degree of freedom in the data as movement.

Figure 11 shows in the first row the ordering of the images along the first principal component calculated by PCA. From the viewpoint of a human observer this ordering does not carry any meaningful insight. The ordering in the second row on the other hand corresponds to a step-by-step rotation of the mobile light source around the ball.

The next experiment introduces two degrees of freedom. We generate 5000 images with resolution 100×50 pixels of a statue (as shown in Figure 12). It is illuminated

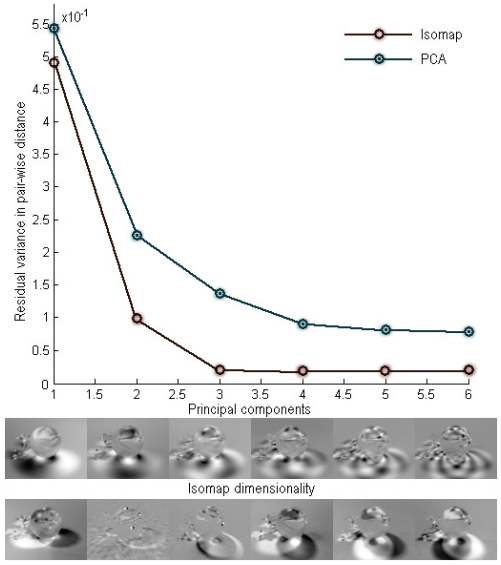


Figure 10: The residual (unaccounted for by the currently used model) variance along the first 6 degrees of freedom detected by PCA and Isomap respectively. The sharp drop in the graph indicates that PCA as well as Isomap has detected the intrinsic dimensionality of the data.

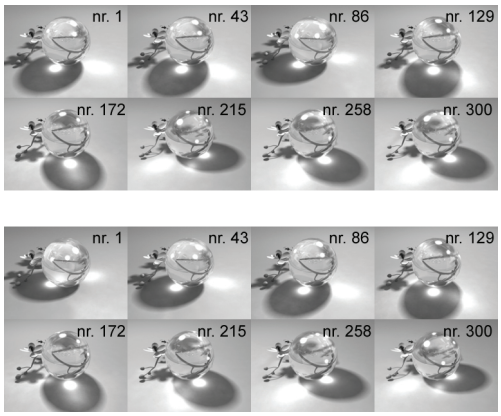


Figure 11: The first group shows the ordering according to the first principal component. The second group shows the ordering according to the first degree of freedom detected by Isomap.

by two light sources, one of which traverses a similar trajectory as in the previous experiment. What also changes in these images is the camera position, which traverses a one-dimensional path folded in three dimensions around the statue.

Similar to the previous experiment, Figure 12 shows the first 6 principal components detected by PCA and the resulting ordering of the data; Figure 13 shows the variations along the degrees of freedom with the 6 highest residuals resulting from Isomap with $k = 10$ nearest neighbors considered in the neighborhood graph, again followed by the resulting ordering of the data. The difference is less obvious this time. For PCA the following observation can be made: the smaller the corresponding eigenvalue, the higher the frequency of the information carried by the principal component. Again there is a significant drop after the

largest residual to indicate a sub-space of the original domain well suited for representing the data, in this case - the overall lightness of the scene. The behavior of Isomap is also similar to that in the previous case with the difference that the sharp drop in the residuals occurs after the second detected degree of freedom and the result gives a human observer information about the parameters responsible for the variation in the image data.

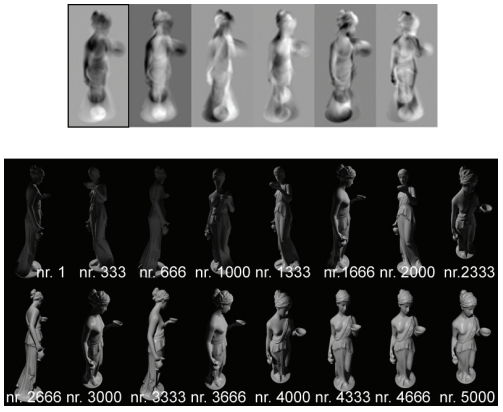


Figure 12: The ordering of the statue images according to the first principal component (highlighted in the top row) detected by PCA.

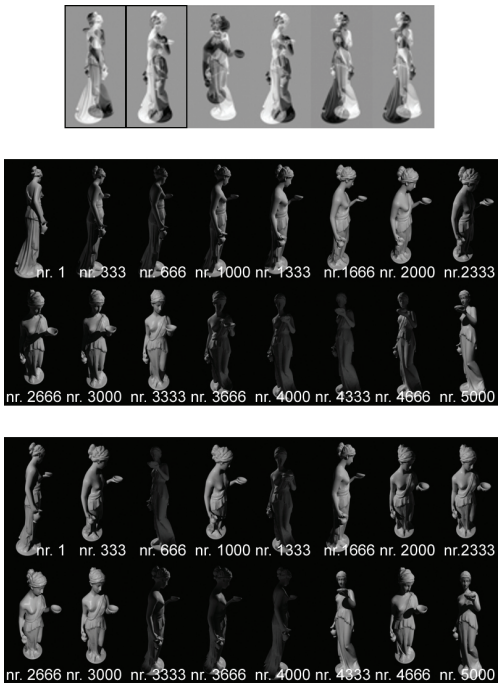


Figure 13: The ordering of the statue images according to the first and second degrees of freedom detected by Isomap: the camera path and the light source path. The variation (gradient) along the first 6 degrees of freedom is in the top row.

The orderings in Figure 13 show a step-by-step traversal of the camera path, and a step-by-step change of the position of the mobile light source. The second ordering is remarkable in its independence from the camera position - the ordering is correct regardless of the overall lightness of the scene.

5 PCA and Isomap Applied to a Natural Dataset

The experiments on natural datasets documented in [5] show that Isomap performs poorer than PCA in four out of five cases. The first reason for that according to Maaten *et. al.* is the 'curse of dimensionality' - the fact that with the rising of the dimensionality of the underlying manifold the number of data samples needed for its proper detection rises exponentially. The second reason is the presence of noise in the natural datasets that results in local overfitting. On the other hand, outliers result in very similar performances. While even a single outlier can cause a 'short-circuit' in the neighborhood graph of Isomaps, the more 'short-circuits' Isomap produces the more the spatial dissimilarity matrix approaches the squared Euclidean distances matrix for the data samples and consequently PCA (see Section 3). Consequently, in the presence of enough outliers Isomap fails to detect non-linear subspaces to the same degree PCA does.

The natural dataset used in this last experiment consists of 1298 archive images with resolution of 410×230 pixels of the maximal walking pressure distribution over a human foot of volunteers aged 12 to 85, courtesy of Sandrina Illes, MSc, TU Chemnitz, Department Forschungsmethoden und Analyseverfahren. The images were obtained by means of a RSScan 0.5 Gait Scientific foot scanner with spatial resolution of 4 sensors per cm^2 and temporal resolution of 300 Hz.

Figure 14: Dataset preparation. The second image from the right represents dataset GS, the rightmost image - the dataset BW.

In the initial stage we prepared the images as illustrated in Figure 14. First the color coded pressure distribution was converted into intensity values. Subsequently all discontinuities were eliminated through linear interpolation in order to improve the conditions for the application of Isomaps (see Section 3). This was followed by the global alignment of all images. The thus prepared grayscale images built dataset GS. The binary dataset BW was derived from GS by employing thresholding followed by a low pass filter to remove noise. In order to reduce dimensionality we scaled the images to a resolution of 64×36 pixels (2304 dimensions). Tests with different did not produce significant changes in the final result.

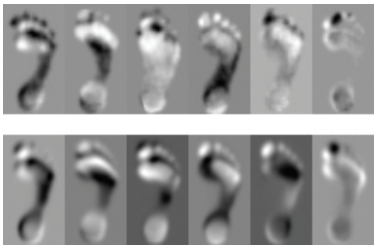


Figure 15: The first row shows the variance along the first 6 degrees of freedom detected by Isomap. The second row shows the first 6 principal components calculated by PCA.

Figure 15 shows the comparison between the first 6 principal components detected by PCA and the variations along the degrees of freedom with the 6 highest residuals resulting from Isomap with $k = 20$ nearest neighbors applied to dataset GS. We performed the Isomap algorithm with values for k ranging from 2 to 50 and chose 20 as it produced the least number of apparent ordering errors. PCA failed to detect any sharp drop in the Eigenvalues corresponding to the first 20 principal components. Isomap exhibited the same behavior in respect to the residuals corresponding to the first 20 detected degrees of freedom.

Figure 16 shows the ordering along the first, second, and third degrees of freedom detected by Isomap respectively. As opposed to the results in the previous section, the interpretation here is difficult due to lack of prior knowledge of the intrinsic dimensionality of the original domain. A tentative interpretation of the results was attempted by an expert in the field of gait analysis, Sandrina Illes, MSc.

The first degree of freedom seems to indicate the amount of pressure on the joint of the big toe. The second degree of freedom seems to correspond to the pressure distribution on the front of the foot, the foot arches, and the heel. The third degree of freedom seems to relate to the pressure on the base of the second and fifth metatarsals. The forth degree of freedom seems very similar to the first one. The fifth degree of freedom seems to indicate the amount of pressure on the toes compared to the pressure on the front of the foot, and the sixth degree of freedom seem to give an ordering according to the presence of Hallux Valgus (bunion).

Figure 17 shows the comparison between the first 6 principal components detected by PCA and the variations along the degrees of freedom with the 6 highest residuals resulting from Isomap with $k = 20$ nearest neighbors applied to dataset BW. We performed the Isomap algorithm with values for k ranging from 2 to 50 and again chose 20 as it delivered the most consistent results. PCA failed to detect any sharp drop in the Eigenvalues corresponding to the first 20 principal components. Isomap exhibited the same behavior with respect to the residuals corresponding to the first 20 detected degrees of freedom.

Figure 18 shows the ordering along the first, second, and third degrees of freedom detected by Isomap respectively. A tentative interpretation of the results was again attempted by Sandrina Illes, MSc.

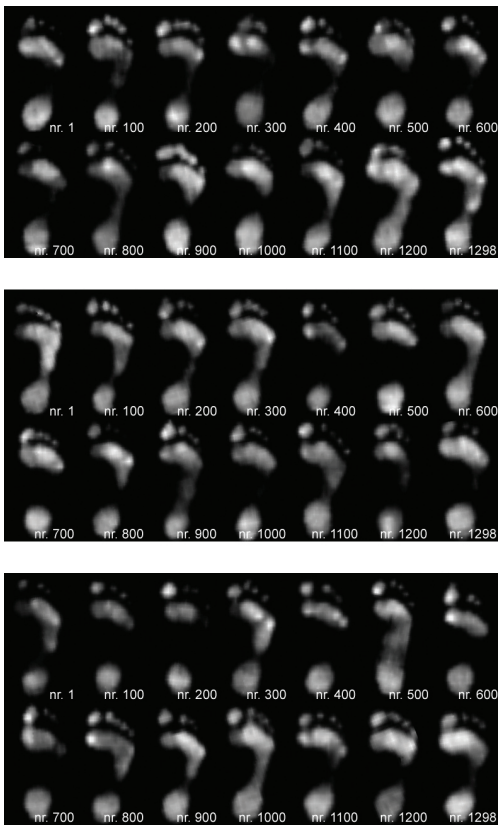


Figure 16: The ordering of the GS foot images according to the first three degrees of freedom detected by Isomap.

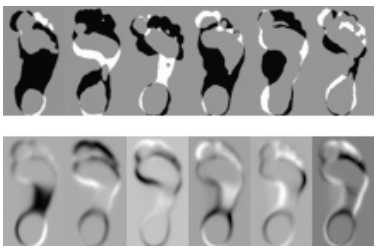


Figure 17: The first row shows the variance along the first 6 degrees of freedom detected by Isomap. The second row shows the first 6 principal components calculated by PCA.

The first degree of freedom seems to produce an ordering from very high to completely collapsed foot arches. The second degree of freedom seems to correspond to the overall shape of the foot with an emphasis on the relation between the length of the longitudinal arch and the foot length. The third degree of freedom seems to relate to the form of the front of the foot and the degree to which the small toes are used. The forth degree of freedom seems somewhat similar to the first one. The fifth degree of freedom seems to indicate the amount to which the transversal arch touches down, and the sixth degree of freedom seems to be in relation to the amount of pressure on the joint of the big toe as well as the function of the other four toes.

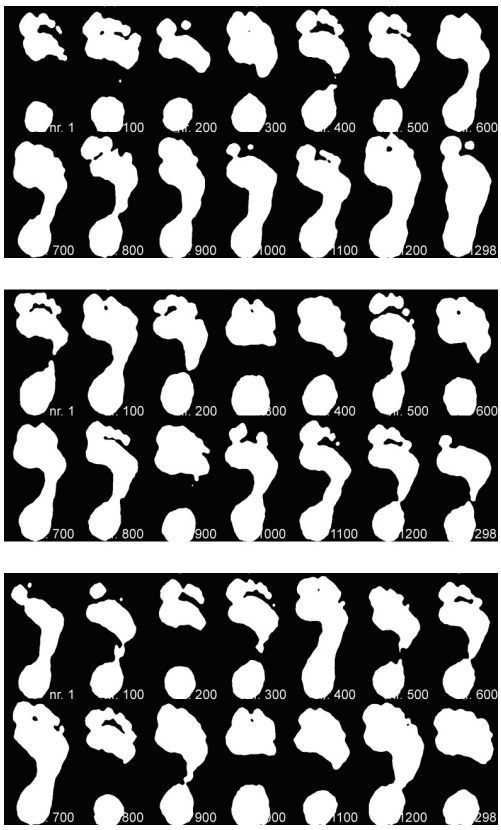


Figure 18: The ordering of the BW foot images according to the first three degrees of freedom detected by Isomap.

6 Results

The results of our experiments in Sections 4 and 5 confirm the performance predictions in [1] and the findings in [5]. Isomap delivers the best results in recognising the intrinsic dimensionality of artificial datasets since they can be sampled adequately due to our a priori knowledge of their structure. The dimensionality of the original data samples does not influence the results. The failure to satisfy the sampling condition (see Section 4.1) results in 'thickened' manifolds [5] due to noise from 'short-circuiting'. Nevertheless Isomap still delivers good results. The failure to satisfy the neighborhood condition however causes convergence of the results from PCA and Isomap - i.e. inability to detect non-linear subspaces in the sampled data.

The main difficulty with natural datasets lies in our lack of prior knowledge. Since we cannot determine if the data samples satisfy the two conditions in Section 4.1 it is highly likely that one or both of the problems described in the paragraph above will occur, which in turn makes interpretation of the result even more challenging.

For the low-dimensional examples in Section 4.1 both PCA and Isomap required less than 1 min. to calculate in $MatLab^{TM}$ on a dual core 1.8 GHz, 2 GB RAM 32-bit Windows platform. For the image examples in Section 4.2 Isomap needed 4 min. for the ball, 10 min. for the foot BW, 12 min. for the foot GS and 20 min. for the statue. Classical PCA needed 6 min. for the ball, 5 min. for the foot BW, 7 min. for the foot GS and 20 min. for the statue datasets. Dual PCA needed less than 2 min. per example.

7 Conclusions

Isomaps is a non-linear method for dimensionality reduction that preserves the geodesic distances between the data samples in their lower-dimensional representation, if such representation can be found. When applied to sufficiently densely sampled artificial datasets it outperforms PCA significantly both in detecting underlying lower-dimensional highly folded subspaces as well as in achieving readability for human observers. On natural datasets PCA and Isomaps deliver comparable results due to our inability to ensure a satisfactory sampling density (at least half of the smallest distance between manifold 'folds') and an adequate neighborhood definition. In order for a human observer to glean new insight in a heretofore unknown domain by means of Isomap one needs to proceed iteratively, adjusting the number of nearest neighbors to be considered in the neighborhood graph and, even more importantly, the data sampling density in each iteration.

References

[1] Mira Bernstein, Vin de Silva, John C. Langford, and Joshua Tennenbaum. *Graph Approximation to Geodesics On Embedded Manifolds*. Stanford University, Palo Alto, CA, USA, December 2000.

[2] Ali Ghodsi. *Dimensionality Reduction a Short Tutorial*. Department of Statistics and Actuarial Science University of Waterloo, Waterloo, Ontario, Canada, 2006.

[3] Christopher Summerfield and Tobias Egner. *Expectation (and attention) in visual cognition*. Trends in Cognitive Sciences, vol. 13, no. 9, pp. 403-409, Sptember 2009.

[4] Joshua Tennenbaum, Vin de Silva, and John C. Langford. *A Global Geometric Framework for Nonlinear Dimensionality Reduction*. Science, vol. 290, pp. 2319-2323, December 2000.

[5] L.J.P. van der Maaten, E.O. Postma, and H.J. van den Herik. *Dimensionality Reduction:A Comparative Review*. MICC, Maastricht University, Maastricht, The Netherlands, January 2008.

[6] Dan Ventura. *Manifold Learning Examples - PCA, LLE and ISOMAP*. Department of Computer Science, Brigham Young University, Provo, UT, USA, October 2008.

[7] R. Viertl. *Einfuehrung in die Stochastik*. Springer-Verlag, Wien, Germany, 2003.

[8] Tynia Yang, Jinze Liu, Leonard McMillan, and Wie Wang. *A Fast Approximation to Multidimensional Scaling*. IEEE workshop on Computation, 2006.

[9] Xu Yaoda and Marvin M. Chun. *Selecting and perceiving multiple visual objects*. Trends in Cognitive Sciences, vol. 13, no. 4, pp. 167-174, April 2009.

Workflow for real-time simulation of deformable objects

Karolina Lubiszewska*
Supervised by: Anna Tomaszewska

Computer Graphics Group
West Pomeranian University of Technology, Szczecin

Abstract

In this paper we address the problem of content creation for physical simulation of soft body objects. We aim to optimize the process of modeling deformable bodies with complex rigid-body skeletons which can be used to visualize realistic movement and animations. Currently no efficient or standarized asset design and implementation method exists for this type of models. We propose reorganization of the present segregation of duties between designer and developer, through a new specialized data interchange file format and the use of extensible open-source designing environment Blender. Simplification of programming work is achieved without unnecessary workload addition for the content creator. Blender is used for object modeling as well as for physical properties and skeleton specification. In result a crucial part of the design workflow is extracted outside of the game engine's SDK toolkit towards independent 3D modeling tools. To evaluate the proposed method a real-time physically-based soft-body character animation is created using Nvidia PhysX and OpenGL.

Keywords: deformable objects, soft body physics, content creation, Blender, rigid skeletons.

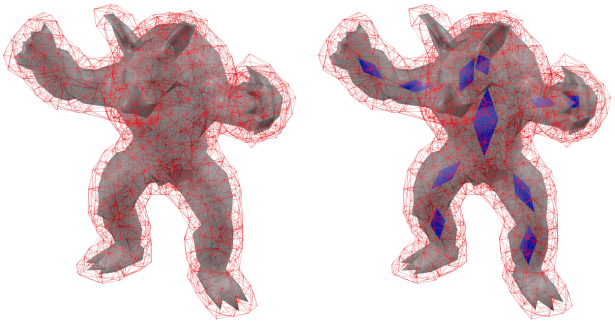
1 Introduction

When rendering real-time computer-generated 3D scenes, the goal is often to create as realistic-looking impression as possible. Under the term of realism we understand 3D models which first of all look life-like, but also behave in a physically-correct manner [4].

Different types of objects behaviour of which can be simulated exist and their simulation requires different approaches. Examples of such objects include:

- rigid bodies - firm, not changing their shape,
- soft bodies - deformable, elastic, fluids, simulating substances like metal, rubber or water.

Depending on the type of given model, for proper simulation of its physical behaviour different kinds of meshes are used. For very simple rigid objects it is enough to provide just the same polygon mesh as the one used for rendering purposes. For more complex objects often a simplified convex hull dedicated to collision detection is supplied. Soft bodies, however, require two different kinds of meshes: one built of planar polygons for rendering purposes, and the other built of tetrahedra (Figure 1a) necessary for the simulation of object's volume.



(a) (b)

Figure 1: Soft body with its tetrahedral volumetric mesh (a) and its counterpart rigged with a skeleton (b).

In the paper we focus on models which consist of both soft and rigid objects (Figure 1b). These are mainly elastic bodies built over rigid-body skeletons. Such complex way of composing and physically simulating objects is not yet popular in video games and real-time virtual reality applications. This situation is mainly caused by the computational cost regarding non-rigid bodies. Every soft object is represented by a number of tetrahedra, each of which has its own position, physical properties and constantly influences the state of every adjacent element. Computations such as the previously described are not required when simulating rigid bodies as their shape is invariable, so is their volume.

Until recently it was impossible to simultaneously simulate and visualize several such objects in real time. Currently thanks to the improving processing power of modern hardware and still improving software tools such as physics and rendering engines, we can finally simulate real soft bodies instead of using keyframe interpolation for smooth animation of deformations [8]. This fact intro-

*klubiszewska@wi.zut.edu.pl

duces completely new possibilities in depicting the reality which surrounds us.

Today even for the simplest models their description is divided into parts. Renderable mesh data and skeleton data are exported to a different file than the tetrahedral mesh necessary for soft body simulation. In addition some information has to be provided manually by the designer in descriptive documents, which are then interpreted by the programmer. A unified, universal and open format is missing.

We propose gathering all the data using an extensible file format to automate and optimize the content creation process and to simplify building simulations which make use of deformable bodies. The goal is to reduce the developer's workload without unnecessarily increasing the number of artist's, designer's and animator's responsibilities by allowing them to work with a single, well-known software suite. Such an optimization will lead to faster, more efficient transferring of created models from design software to end applications. It is most important for video games asset creation, where artist's fantasy is one of the key aspects leading to success.

In the next chapter we provide a short survey of existing approaches to the simulation of soft bodies, or methods which can be used in replacement. The third chapter covers the issues related to the current workflow and proposes our solution to the outlined problems. In the fourth chapter we present implementation-related details important when using our method. We conclude the paper with results gathered from building a test scene containing a soft body with a rigid skeleton and we compare it with a sole deformable body. In the end our future goals are described.

2 Related work

The idea of soft body simulation for computer graphics applications was proposed in the late '80s [15]. The first trials were conducted with non-real-time simulations only as real-time visualization of complex objects was far out of the scope for that time hardware.

There are many different approaches for simulating soft bodies. They were comprehensively surveyed in [5] and [10]. We have chosen the method which bases on using tetrahedral lattice mesh for representing the object's volume. The mesh consists of finite number of elements which thoroughly fill the modeled object's extent [14]. Its vertices form the topology of a mass-spring system used for simulating the body's interaction and self collisions [2]. Cost of this approach can be easily scaled to suit the needs of a particular model, scene and hardware capabilities. It makes the method useful for interactive simulations.

To address the calculation complexity related issues, a keyframe-based simplification can be used. The actual deformations are calculated off-line and only certain keyframes are exported for use in real time. These keyframes are then interpolated to resemble smooth animation [8]. This solution lacks the freedom of interaction as the object's reactions are limited only to a strictly defined set of previously prepared possibilities. To achieve the true real-time simulation we decided to perform online calculations for the whole object's volume instead of using the keyframes.

However, a body which only consists of a deformable volume is hardly controllable and classic methods for e.g. character animation cannot be used. To solve this problem a coupling between the soft body and a rigid skeleton can be used. The skeleton can react to applied forces and movement induced by methods such as reverse kinematics [7]. The use of a skeleton allows us to introduce limits for bone joints which restrict the movement that could be recognized as unnatural depending on the object's characteristics [6, 13].

The field for soft bodies application, including those equipped with a rigid skeleton, is very broad. Apart from the already mentioned character animation, deformable objects are used for modeling destruction [12] and jelly-like entities [3] in video games. Moreover physically-correct simulation of soft tissues is pursued by the numerous virtual reality applications aimed at training medical personnel [1, 9].

Using a soft body in real-time graphics visualization presents many benefits:

- preserves a consistent volume - as its finite elements influence each other,
- smooth deformations can be applied,
- is elastic - retains its previous form,
- tearing can occur - in effect it breaks the former topology and forms two or more objects.

3 Proposed content creation workflow

In order to simulate a soft body together with its rigid skeleton, appropriate input data are needed. These include not only the renderable mesh information, but purely physical properties as well. The following are necessary:

- coordinates of polygonal mesh vertices for rendering purposes,
- coordinates of tetrahedral mesh vertices for volume simulation,
- coordinates of rigid body vertices,
- information on joints and their types for coupling rigid bodies,
- limitation and spring information for joints,

- mass (for both body types),
- friction (for both body types),
- gravity response (for both body types),
- texture coordinates for renderable soft body mesh,
- information necessary for dynamic calculation of normal vectors.

3.1 Current content creation workflow

On the diagram (Figure 2) the problem of current workflow for designing soft body with rigid skeleton is pictured.

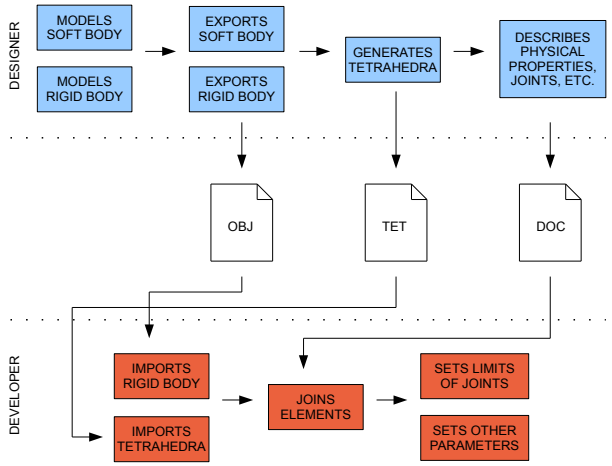


Figure 2: Current content creation workflow for designing soft body with rigid skeleton.

It can be seen that the pipeline is a multi-stage, complex process which can result in numerous misunderstandings between the design and development teams.

Currently the artist who designs the 3D model creates a mesh for both soft body and rigid skeleton. Then the parts are exported to appropriate files. Additionally he is responsible for creation of volumetric tetrahedral mesh in a separate, specialized program in order to allow the deformation of soft body structure. The next step is the description of dependencies between certain objects, the physical properties, joints etc. to achieve the desired object rigging.

For example, an elbow exposes a naturally limited freedom of swinging angle and direction. The joint prevents the bones from moving away from each other (Figure 3). These limitations have to be applied to simulated bones as well. And because of no popular, cross-platform solution, the risk of misunderstandings between cooperating persons is high.

The developer receives a set of files that have to be imported one-by-one in the end application. The additional provided information has to be implemented manually, the joints need to be set up, the physical properties and the

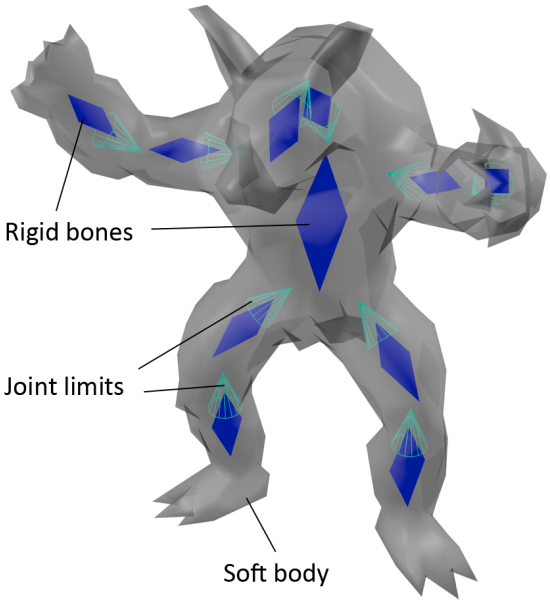


Figure 3: Stanford Armadillo skeleton with joint limits visualized as cones.

joint limitations need to be introduced according to the designer's description.

The current workflow requires the developer to possess knowledge about the model which he imports. He has to perform numerous object-dependent actions manually. The complexity of the workflow, its time-consuming aspect and proneness to mistakes does not allow for efficient use in the creation of deformable assets for real-time computer graphics applications.

3.2 Solution

The diagram (Figure 4) presents our proposed content creation pipeline for designing soft body with rigid skeleton.

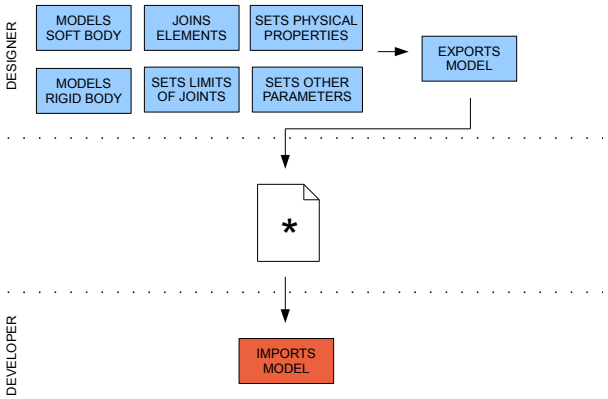


Figure 4: Proposed content creation workflow for designing soft body with rigid skeleton.

The main aim of the improvements is to reduce the programmer's workload. As the diagram pictures, the most

of the stages are now controlled by the designer. However, despite controlling the major part of the workflow, the designer is not overwhelmed with responsibilities. He describes the object's properties within the same software suite he uses for modeling. The programmer's only remaining duty is to import the resulting file to make the object available for simulation and rendering.

During the first stage the 3D artist creates meshes for objects: deformable one related to the body silhouette and rigid ones for the skeleton. Then he is able to create joints between the skeleton parts and set their limits to form the object's rigging. Physical properties are assigned to proper parts of the model and other rendering-related data such as the material and texture data can be applied as in regular objects modeling pipeline. While exporting, the tetrahedral mesh is generated automatically for the deformable volume.

Differences between the proposed and current solution are relatively insignificant from the designer's point of view, but at the same time the programmer is left with significantly less responsibilities. Instead of descriptive documentation, the artist can configure properties already in the design software (Figure 5). Finally a single file is exported for the complete complex model.

The programmer imports received data to the final application using the importer described later in chapter 4. He is not obliged to specify the parameters, join specific objects and set the limits by himself. He does not need to know anything specific about the particular model as during the import procedure all the necessary data are processed automatically. The imported model is ready-to-use.

Our solution helps the content creation automatization and reduces the necessary workload. It also eliminates the chance for misunderstandings resulting from descriptive documents.

4 Technical details

In order to test our solution we created a simple application which generates a real-time animation of Stanford Armadillo model. Our object consists of a soft body rigged with a rigid-body skeleton. Armadillo's arms are attached to invisible blocks floating in space. Bones of the skeleton are coupled using 3-degrees-of-freedom spherical joints with limited ability to swing so that the character cannot hang limply. The model is influenced by gravity and objects which can be thrown at it to visualize the deformations.

For 3D object design purposes we use *Blender 2.61* (Figure 5) software which is freely available and allows us to create our own plug-in extensions. Visualization is performed in real time by a simple *OpenGL 2.1* and *Glut* library-based rendering engine written in *C++*. *NVIDIA PhysX 2.8.1* engine is responsible for physics-related calculations and soft body simulation.

Currently we use *PhysX Viewer* which is a part of the *NVIDIA PhysX SDK* to create a tetrahedral mesh. However in the future the lattice is to be created automatically during exporting procedure of the model from design environment. We plan to use the algorithm described in [14] for this purpose.

Figure 5: Creating model skeleton of Stanford Armadillo in Blender 2.61.

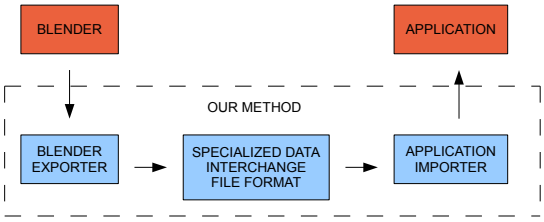


Figure 6: Implementation schema using proposed tools.

Exemplary contents of an *XML* file are shown in Figure 8. They include the contents of both standard *Wavefront OBJ* and native *NVIDIA PhysX TET* files. The file contains skeleton's polygon meshes, tetrahedral volumetric lattice mesh, renderable soft body surface polygon mesh, descriptions of skeleton joints and various physical properties of both rigid and soft parts. We use *XML* mainly to depict the hierarchy of elements which are parts of the proposed format. In the future a binary file format should be considered for final implementation to reduce the file size and improve the processing speed during import.

The whole structure is divided into parts which relate to soft body and rigid bodies.

In the soft-body section there are:

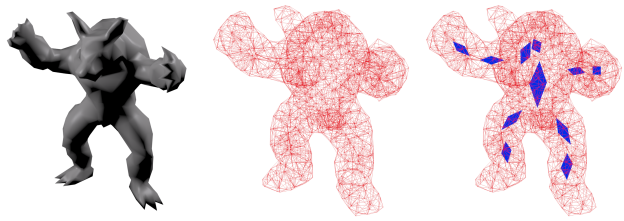
- tetrahedral mesh which describes the volume:
  - vertices
  - configurations of consequent tetrahedra
- polygon mesh for rendering purposes:
  - vertices
  - texture coordinates
  - configurations of consequent polygons

- barycentric coordinates - mapping of renderable mesh vertices into the volume of a specified tetrahedron which includes the vertex in the initial pose
- physical attributes:
  - mass
  - volume stiffness
  - stretching stiffness
  - friction
  - particle radius, solver iterations - scene-dependent values important for simulation stability

The rigid-body-skeleton section consists of:

- convex polygon meshes for collision detection:
  - vertices
  - configurations of consequent polygons
- physical properties:
  - mass
  - friction
- joints between rigid bones:
  - type-dependent values and limitations

When the model is imported, information chunks are routed to appropriate application elements: the rendering engine and physics engine.



(a) (b) (c)

Figure 7: Renderable surface mesh (a), tetrahedral lattice mesh (b), tetrahedral mesh with rigid skeleton (c).

Polygon surface mesh for the soft body (Figure 7a) together with its texture coordinates is passed directly to the rendering engine. The tetrahedral mesh (Figure 7b) is passed to *NVIDIA PhysX*, same as the convex skeleton meshes (Figure 7c). In our case the skeleton is intended for physics simulation only, it is not intended for drawing.

For every rigid bone a separate *actor* is created in *PhysX*. Each of them is coupled with other bones using a joint according to the joints section of model file. The joint can be any of the 10 types offered by *NVIDIA PhysX* [11], e.g.:

- Spherical joint;
- Revolute joint;
- 6-degree-of-freedom joint;
- Distance joint.

Every joint can have different limitations depending on its type. For example, the spherical joint can be restricted to swing or twist only in a specified angle range.

For the soft body no actor is created. Instead, an instance of a specialized *PhysX* soft body class is used. It receives the necessary parameter values from the imported model file, such as mass, stiffness and friction.

To visualize the simulation, during every animation frame the position of each renderable surface mesh vertex is updated according to the simulation results. The update is performed with the help of four-dimensional barycentric coordinates B_{V_i}, which are defined for every vertex V_i of the polygon surface mesh:

$$B_{v_i} = (v_0, v_1, v_2, 1 - (v_0 + v_1 + v_2))$$

Where the k-th element of the vector B_{V_i} we write as $B_{(V_i,k)}$. The coordinates are calculated for the initial pose of renderable mesh vertices in the volume built of tetrahedra. To obtain the current vertex position V_i' necessary to visualize the soft body, we calculate an affine combination using current position P' of vertex T_j of the tetrahedral mesh:

$$V_i' = \sum_{k=1}^{4} B_{(V_i,k)} P'_{(T_j,k)}$$

In order to calculate correct lighting of the soft body, in every frame we use the already updated positions of vertices to calculate current normal vectors N_l for every l-th face. It amounts to applying a cross product between two edges E_1 and E_2 of a given triangle built from vertices V_0, V_1 and V_2.

$$E_1 = V_0 - V_1$$

$$E_2 = V_2 - V_1$$

$$N_l = E_1 \times E_2$$

The order of vertices depends on the order in which the triangles were defined: clockwise or counter-clockwise. To achieve smooth shading, per-vertex normal vectors have to be averaged and normalized.

The soft body can be attached to its skeleton in two ways:

- one-by-one explicitly specified rigid objects are attached to the soft volume,
- every rigid object which collides with the volume is automatically attached.

Advantage of the first approach is that the coupling can be strictly defined. In the second case the attachment procedure has to take place in a controlled space, where no other object can appear by accident. Otherwise unwanted attachment can occur. But when using this way no additional data and activity is necessary for creating the coupling.

5 Results

The resulting visualization of rigid-skeleton-rigged soft body which makes use of joint limits behaves much more naturally than when not using the skeleton (Figures 9, 10). It is worth noticing that the limbs of Stanford Armadillo do not bend under angles which could look unrealistically. Without the skeleton there is no such restriction and the deformation can occur in a completely random manner. The other advantage is the ability to control the soft body with inverse kinematics, what is impossible in a pure-soft-body solution. Also the ability to set different masses to different bones in order to achieve non-uniform mass distribution improves the realism of character's reactions.

In our example the volumetric lattice consisted of approximately 2750 tetrahedra. The renderable surface mesh contained 1036 polygons. The low number of polygons resulted from the necessity to recalculate vertex positions and normal vectors every frame what makes achieving interactive frame rate challenging. While rendering two such models simultaneously on the scene we reach 18 frames per second on an *NVIDIA GeForce GTX 295* and *AMD Phenom II X4 965*.

We observed during the implementation process that it is important to match the soft body volume's particle radius with the simulation conditions like the model's size and density of the lattice mesh. Otherwise skeleton attachment can be problematic as some parts of the bones can slip in-between the particles. It is also important not to forget about setting friction high enough to avoid simulation instability. The importance increases with the particle radii as the particles begin to constantly influence each other.

6 Summary and future work

We presented a file format and workflow improvements which can lead to enhanced work efficiency during design phase of rigid-skeleton-rigged soft body simulations. Behaviour of the model which is simulated with a skeleton proved to be closer to the expected and natural than when using a sole soft body or a rigid body instead. We believe that in the future video games will benefit from using this approach for character animation instead of the currently common unrealistic solutions.

One of our future goals is to create a Blender plug-in for exporting models to our universal format (Figure 8). The exporter should allow easy definition of different physical properties and intuitive coupling of skeleton parts together with setting their attributes and limits using the Blender built-in armature interface. Also the automatic generation of tetrahedral mesh should be introduced to the exporter. Our file format should be extended to cover other important rendering-related features such as materials and textures. Interesting soft-body features such as tearing and heterogeneous materials stay in our field of interest as well.

References

[1] H. Delingette. Toward realistic soft-tissue modeling in medical simulation. *Proceedings of the IEEE*, 86(3):512–523, 1998.

[2] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 1–8, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[3] Epic Games, Inc. Unreal Engine 3 UDK: PhysX soft bodies. http://udn.epicgames.com/Three, 2010.

[4] J. Ferwerda. Three varieties of realism in computer graphics. In *SPIE Human Vision and Electronic Imaging 03*, pages 290–297, 2003.

[5] N. Galoppo. *Animation, simulation, and control of soft characters using layered representations and simplified physics-based methods*. PhD thesis, Chapel Hill, NC, USA, 2008. AAI3331034.

[6] J. Georgii, D. Lagler, C. Dick, and R. Westermann. Interactive deformations with multigrid skeletal constraints. In Kenny Erleben, Jan Bender, and Matthias Teschner, editors, *VRIPHYS*, pages 39–47. Eurographics Association, 2010.

[7] J. Kim and N. Pollard. Fast simulation of skeleton-driven deformable body characters. *ACM Trans. Graph.*, 30:121:1–121:19, October 2011.

[8] R. Kondo, T. Kanai, and K. Anjyo. Directable animation of elastic objects. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 127–134, New York, NY, USA, 2005. ACM.

[9] A. Maciel, T. Halic, Z. Lu, L. Nedel, and S. De. Using the physx engine for physics-based virtual surgery with force feedback. *The international journal of medical robotics computer assisted surgery MRCAS*, 5(3):341–353, 2009.

[10] A. Nealen, M. Mller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models

in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[11] NVIDIA. PhysX SDK 2.8 introduction. http://developer.nvidia.com, 2008.

[12] E. Parker and J. O'Brien. Real-time deformation and fracture in a game environment. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 165–175, New York, NY, USA, 2009. ACM.

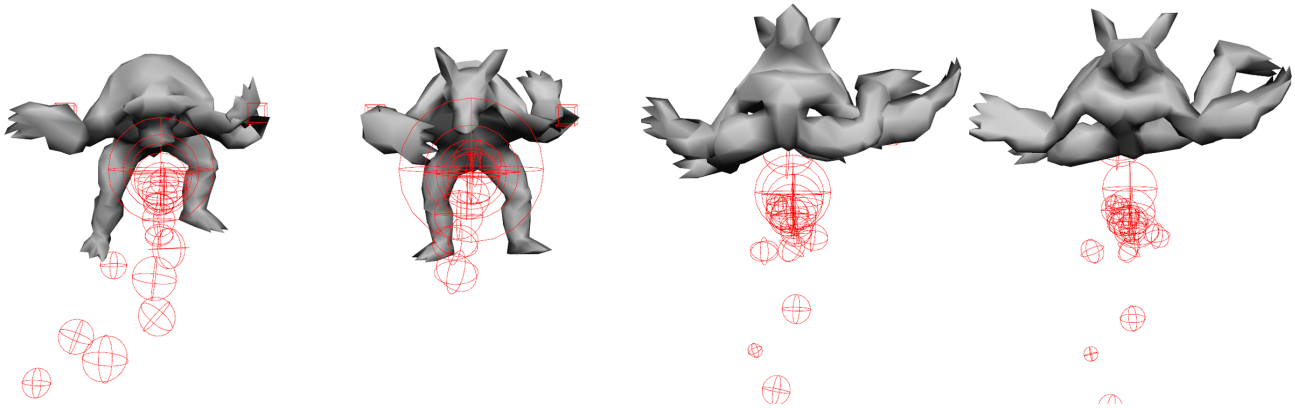
[13] T. Shinar, C. Schroeder, and R. Fedkiw. Two-way coupling of rigid and deformable bodies. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 95–103, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[14] J. Spillmann, M. Wagner, and M. Teschner. Robust tetrahedral meshing of triangle soups. In *Vision, Modeling, Visualization (VMV)*, pages 9–16, 2006.

[15] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 205–214, New York, NY, USA, 1987. ACM.

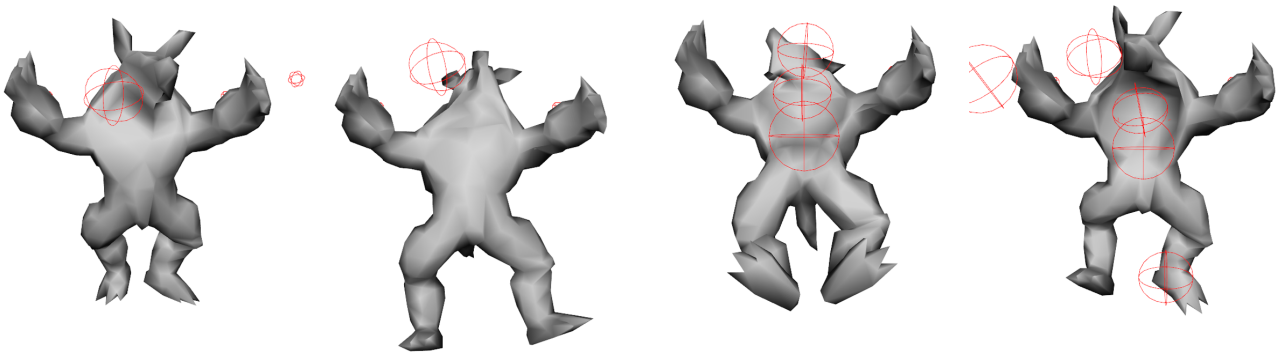
```
<?xml version="1.0" encoding="utf-8"?>
<model>
  <soft_body>
    <tetrahedron_mesh target="volume">
      <vertices>
        <vertex id="1" x="8.4" y="2.3" z="9.5"/>
        <!-- ... -->
      </vertices>
      <tetrahedra>
        <tetrahedron>
          <node v="1"/>
          <!-- ... -->
        </tetrahedron>
      </tetrahedra>
    </tetrahedron_mesh>
    <polygon_mesh target="rendering">
      <vertices>
        <vertex id="1" x="8.4" y="2.3" z="9.5"/>
        <!-- ... -->
      </vertices>
      <texture_coords>
        <coord id="1" u="8.4" v="2.3"/>
        <!-- ... -->
      </texture_coords>
      <normals>
        <vector id="1" x="8.4" y="2.3" z="9.5"/>
        <!-- ... -->
      </normals>
      <faces>
        <face>
          <node v="1" n="44" t="45"/>
          <!-- ... -->
        </face>
      </faces>
    </polygon_mesh>
    <barycentric>
      <coord render_id="7" tetr_id="3" v0="0.2"
       v1="0.2" v2="0.1"/>
    </barycentric>
    <attributes>
      <mass>1.0</mass>
      <volume_stiffness>0.5</volume_stiffness>
      <stretching_stiffness>0.9</stretching_stiffness>
      <friction>0.9</friction>
      <particle_radius>0.4</particle_radius>
      <solver_iterations>10</solver_iterations>
    </attributes>
  </soft_body>
  <rigid_bodies>
    <rigid_body id="1">
      <polygon_mesh target="collision_detection">
        <vertices>
          <vertex id="1" x="8.4" y="2.3" z="9.5"/>
          <!-- ... -->
        </vertices>
        <faces>
          <face>
            <node v="1"/>
            <!-- ... -->
          </face>
        </faces>
      </polygon_mesh>
      <attributes>
        <mass>1.0</mass>
        <friction>0.9</friction>
      </attributes>
    </rigid_body>
    <!-- ... -->
    <joints>
      <spherical_joint r1="1" r2="2">
        <limits>
          <limit target="swing" angle="45"/>
        </limits>
      </spherical_joint>
      <!-- ... -->
    </joints>
  </rigid_bodies>
</model>
```

Figure 8: Exemplary model file contents for a soft-body object with a rigid skeleton.



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

Figure 9: Frames from the animation depicting throwing balls at a pure skeleton-free soft body. Worth noticing is the unnatural bending of limbs in images (b), (c), (d) and the overall inertia resulting from uniform mass distribution.



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

Figure 10: Frames from the animation of a soft body rigged with a rigid skeleton. The shape is retained much more firmly and the character does not expose unnatural behaviour.

Multiview Normal Field Integration using Graph-Cuts

Aljosa Osep*
Supervised by: Michael Weinmann,† Reinhard Klein‡

Institute of Computer Science II - Computer Graphics
University of Bonn
Bonn / Germany

Abstract

While there are many algorithms which address the reconstruction of partial surfaces from single-view normal fields, to the best of our knowledge, there is only one method [Chang et al., 'Multiview Normal Field Integration using Level Sets', CVPR 2007] which focuses on the reconstruction of the full shape of an object from multiple normal fields captured from multiple viewpoints. In this paper, we propose an alternative approach for the integration of such normal fields. We use a similar energy formulation as Chang et al., but replace the employed level-set representation with a graph-cut based reconstruction. Based on the visual hull of the object we estimate the visibility from each viewpoint. Then we project estimated normal fields to the visual hull and the optimal surface is computed by maximizing the flux of the obtained vector field through the surface. The graph-cut approach allows for a fast and globally consistent optimization of the given problem. Finally, we demonstrate the validity of our algorithm on synthetic data sets.

Keywords: 3D reconstruction, normal field integration, graph-cuts, multiview vision

1 Introduction

The digitization of 3D objects is a very important topic in computer graphics and computer vision and requires a faithful reconstruction of the object surface. There are various applications to surface reconstruction of real models in industry, archaeology and art, medical imaging and entertainment. The aim of surface reconstruction is to construct an as accurate as possible approximation of the geometry of a real surface.

For this purpose, many different techniques have been developed. Some methods, such as laser scanning, structured light systems and multiview stereo methods directly reconstruct an oriented point cloud, from which a closed surface can be derived by applying one of the surface fitting methods [13, 4, 20, 18, 12]. In contrast, there are also approaches that only rely on information about the surface normals. Methods such as Shape-from-Shading [23] and Shape-from-Specularity [6] can be used for obtaining a normal field and subsequently, normal field integration methods are applied for estimating the shape of the observed object. However, most of the works address only reconstruction of a certain part of the object surface from the estimated normal fields as they only use a single viewpoint. In this paper, we address the integration of such estimated normal fields from multiple views in order to recover the full 3D shape of the object. The fusion of such information from multiple viewpoints is a challenging problem, because we do not have any spatial surface samples but only normal samples.

To the best of our knowledge, this problem has only been addressed in [5]. The authors derive an energy functional consisting of a surface and the flux term, and minimize it using level-sets. Our work is based on the minimization of the same energy functional. However, we take a different optimization approach. Similarly as in [5], we first compute the visual hull of the object. Based on that we compute an approximation of the visibility. Then, we project the observed normal fields from each view to the visual hull, taking into the account the visibility. We simultaneously optimize the flux through the surface by maximizing divergence and enforce a minimal surface using Graph-Cuts.

The rest of our paper is organized as follows. In the next section, we discuss previous and related work in the area of 3D surface reconstruction using normal field integration. In Section 3, we state our goal more precisely and introduce necessary notations. In Section 4, we explain all steps of our algorithm in detail and in Section 5 we discuss the results of our multiview normal field integration algorithm. Finally, we conclude the paper in Section 6 and discuss possibilities for future work.

2 Previous Work

As pointed out in [11], there is no algorithm, which is able to reconstruct a general scene with any type of materials and lighting conditions. The different approaches are usually designed according to the requirements in the

*osep@informatik.uni-bonn.de
†mw@cs.uni-bonn.de
‡rk@cs.uni-bonn.de

observed scene. For this, they rely on exploiting different visual cues such as silhouettes, textures, shading, specularities, etc.

Assuming we already have normal information as input, we concentrate on reviewing techniques using such normal information to reconstruct a surface. There exists a variety of techniques addressing the normal field integration problem. This is a challenging problem, because in the presence of noise, the observed normal fields are not integrable. For a vector field to be gradient of the function, its curl must be zero, which is rarely the case in the presence of noise and outliers.

Most of the methods addressing this issue attempt to enforce integrability. In [8], the authors project the gradient field to integrable functions using the Fourier basis functions. There also exist several variants of this method, where different basis functions (cosine basis [9], shapelets [16], etc.) are used. Another approach, proposed in [22], attempts to solve the problem by finding the function whose gradient is closest to the observed normal field in the L_2 norm sense by solving the Poisson equation. The authors of [21] observed, that methods based on minimizing least-squares cost functions cannot handle outliers well and propose a method that minimizes an energy functional in the L_1 norm sense. However, all of these methods only address the reconstruction of partial (2.5D) surfaces from a single-view normal field. A detailed review of the related work on single-view normal field integration can be found in [23].

Combining normal field information from several views is addressed in [5]. There, the authors propose an efficient algorithm based on energy minimization, to which our approach is closely related. To the best of our knowledge, their algorithm is the first one that is able to recover the full 3D shape of an object from multiple normal fields. In their work, they derive a geometric PDE that minimizes an energy functional which is composed of mean curvature and flux term. The PDE is optimized using a level set method. This optimization process is a drawback of their approach, since it can find only a local minimum of the energy functional and it is highly dependent on the initial surface.

There are also certain similarities of the considered reconstruction problem to surface reconstruction from oriented point clouds. Among the implicit function fitting-based methods, there are approaches [13, 4, 18] that consider an oriented point cloud simply as a vector field that is in fact a sparse sampling of a continuous vector field corresponding to the true surface ∂M of the solid M. Based on that observation, these methods attempt to find an implicit function $f(x)$ whose gradient $\nabla f(x)$ is as close to the observed vector field $\vec{V}$ as possible. Our reconstruction method is also based on these ideas and observations.

3 Problem Statement

Our goal is to reconstruct the full 3D, closed and twice differentiable surface $\partial M \in \mathbb{R}^3$ of a solid M, given the observed normal fields (normal samples) from different cameras. Furthermore, we want to obtain a watertight surface in form of a polygonal mesh. Note, that in our formulation, we do not have samples of surface points but only samples of their normals, projected to the image planes of the cameras. In our setup, we assume having N calibrated cameras $C_i, i \in [1...N]$, as visualized in Figure 1. Each camera provides a noisy normal field estimate v_i. The mapping $v_i : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{R}^3$ is a projection of normals of the points, seen from the camera C_i to its image plane. For the cameras, we assume a pinhole camera model and we assume that we have for each camera a projection matrix $P_i = K_i\,[R_i|t_i]$. The matrix K_i is the camera calibration matrix and it provides the intrinsic parameters of the camera. The matrix $[R_i|t_i]$ provides information about external camera parameters (position and orientation in space). The projection of the point $x \in \partial M$ to the image plane of i-th camera C_i will be denoted by $\widetilde{x_i} = P_i x$ and $v_i(\widetilde{x_i})$ denotes the projection of the normal $n(x)$ of the point $x \in \partial M$ to the image plane of camera C_i.

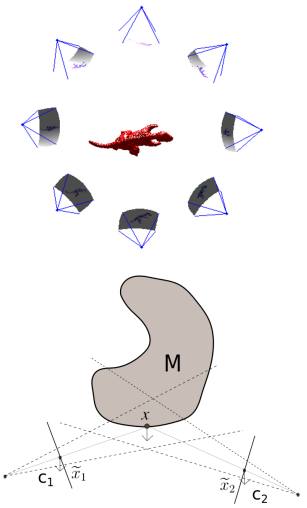


Figure 1: A typical scene setup *(top)*. Projection process *(bottom)*. The surface point x of the solid M is visible in cameras C_1 and C_2 and its normal is projected to their image planes.

We consider a very similar projection process as the authors of [5], which is illustrated in Figure 1. Taking into account the projection process, additive noise and visibility constraints, our normal field formation model can be expressed by:

$$v_i(\widetilde{x_i}) = n(x) + \eta_i, \quad \forall x \in \left\{x | \psi_{i,\partial M}(x) = 1\right\} \qquad (1)$$

where $n(x)$ is normal of the point x, η_i is the corresponding

1. Compute the vector field $\vec{V}$
   (a) Initialize the model by silhouette carving
   (b) Compute the visibility
   (c) Project normal fields
2. Compute the divergence of $\vec{V}$
3. Construct a graph
   (a) Establish *n-links* (Adjacent nodes)
   (b) Establish *t-links* (Terminals)
4. Compute the Min-Cut on the constructed graph using the TouchExpand algorithm [18]
5. Extract the isosurface using Marching Cubes

Table 1: The main steps of our algorithm.

additive noise and

$$\psi_{i,\partial M}(x) \doteq \begin{cases} 1, & x \text{ is visible from the camera } C_i \\ 0, & \text{else} \end{cases} \quad (2)$$

is a visibility function that indicates whether the point x is visible from the camera C_i. Like in [5], we also address inferring the true coordinates of all points $x \in \partial M$ from estimated noisy normal fields and reconstructing the full 3D solid M as accurate as possible.

4 Proposed Algorithm

The problem domain is discretized on a regular grid. Then, the vector field $\vec{V}$ is computed and the energy functional is transformed to a graph. Subsequently, we compute the surface ∂M as a Min-Cut on the energy graph. The polygonal mesh is obtained using the Marching Cubes algorithm [19]. The main steps of our algorithm are listed in Table 1 and detailed explanations of all steps are provided in the following subsections.

4.1 Energy Minimization Framework

Energy minimization is a very popular approach and often used for surface reconstruction, both in the field of single-view normal field integration [23] and surface reconstruction from point clouds [13, 4, 18]. In the first paper (to the best of our knowledge) addressing the problem of multi-view normal integration [5], the problem is formulated in terms of energy minimization as well. Our formulation is based on the very similar energy functional.

The authors of [5] are motivated by the approach described in [22], where a relief surface is reconstructed from a single normal field in the following way:

$$E(Z) \doteq \int_Z (\nabla x - v(\widetilde{x}))dA \quad (3)$$

where the integral is over the planar (image) domain Z. Intuitively, the energy functional penalizes discrepancy between the gradient of the surface we would like to obtain and the observed normal field. The authors of [5] extend the idea to the domain of the surface ∂M. Taking into account the visibility constraints, their energy functional is as follows:

$$E(\partial M) \doteq \int_{\partial M} \frac{1}{N_{\partial M}(x)} \sum_{i=1}^{N} \psi_{i,\partial M}(x) \left\| n(x) - v_i(\widetilde{x_i}) \right\|^2 \mathrm{dA} \quad (4)$$

where the term

$$N_{\partial M}(x) \doteq \sum_{i=1}^{N} \psi_{i,\partial M}(x) \quad (5)$$

denotes the number of cameras in which the surface point x was seen. The proposed energy functional minimizes differences between the normals $n(x)$ of the surface points $x \in \partial M$ and all observed normal samples, from all cameras. The normal sample of the point x, seen by the camera C_i, corresponds to the normal of the projection of x to the i-th image plane. Additionally, for each camera, we take into account only the visible points and we normalize the total deviation by the number of cameras the point was seen from.

Alternatively, we are again looking for the surface whose gradients match best to the observed normals in a least squares sense under the visibility constraints. The authors of [5] observed, that minimizing Equation (4) is equivalent to maximizing the flux through the surface and simultaneously minimizing the surface area. Hence, we obtain a similar energy functional as proposed in [18], which consists of a data term and a regularization term:

$$E(\partial M) = \lambda_1 R(\partial M) - \lambda_2 D(\partial M). \quad (6)$$

Here,

$$R(\partial M) = \int_{\partial M} \mathrm{dA} \quad (7)$$

denotes the area over the surface, and

$$D(\partial M) = \int_{\partial M} n(x)\vec{V}(x)\mathrm{dA}, \quad (8)$$

describes the flux through the surface ∂M. Considering the visibility from each camera, the vector field $\vec{V}$ can be computed according to

$$\vec{V}(x) \doteq \frac{\sum_i \psi_{i,\partial M}(x) v_i(\widetilde{x})}{N_{\partial M}(x)}. \quad (9)$$

The implementation of the vector field computation will be discussed in the next section.

Additionally, we weight both terms. Intuitively, by minimizing this energy functional, we are aligning the surface ∂M with the vector field $\vec{V}$, and with the regularization term, we avoid over-fitting and make the optimization process robust to noise and outliers.

4.2 Computation of the Vector Field

The computation of the vector field $\vec{V}$, as stated in Equation (4), requires summing over all observed normal fields while accounting for visibility and occlusion. That is a non-trivial task, since we do not know where the true surface ∂M lies, i.e. we do not have any information about the real position of the surface points. We approach the

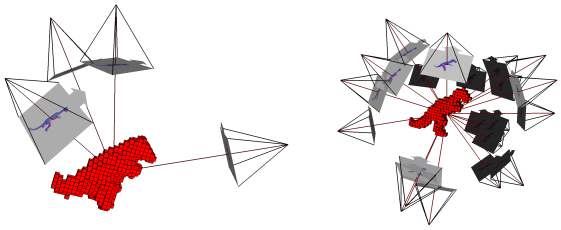


Figure 2: A silhouette carving process. Result of carving using 4 cameras *(top left)* and result using 20 cameras *(top right)*.

problem by exploiting an additional visual cue about the geometry of our solid M that the estimated normal fields provide. To be exact, we compute the visual hull of the object based on the silhouettes obtained from the normal fields [17]. The silhouette carving process is visualized in Figure 2.

We discretize our domain of interest using a regular grid. For this, we first extract the area of interest by intersecting the volumes of all cameras. Then we narrow down this area of interest by performing an initial carving. Doing so, we obtain a rough carved object. We take the maximal and minimal point of this object and we use them to initialize the final bounding volume, on which we perform a fine carving.

Based on the visual hull, which is computed on the initialized voxel grid, an approximation of the visibility function can be computed. Note, that in order to compute the exact visibility function and account for all occlusions precisely, we would need the actual surface ∂M, which is just what we want to obtain. The approximate visibility for the camera C_i is computed by casting rays from the focal point of C_i to the centres of all voxels in the voxel grid. The visibility is determined by intersecting each ray with a triangular mesh representing the visual hull, obtained by Marching Cubes [19]. The location of the exact surface ∂M is not known at this stage, but we know that it must be somewhere close to the visual hull. Furthermore, we know that the actual surface can only be smaller than the visual hull. For that reason, we do not only consider border voxels as visible, but we rather take into account visible bands of voxels, as shown in Figures 3 and 4. A visible band consists of the border voxels, directly visible from camera C_i and the voxels that are for a band-depth ε away from the border voxels towards inner region of the model in the direction of the rays. The effects of the parameter ε will be discussed in Section 5.

Figure 3: Slice of the computed visibility on the voxel grid from the i-th camera. Dark (blue) voxels correspond to the visual hull and bright (green) voxels are visible from camera C_i.

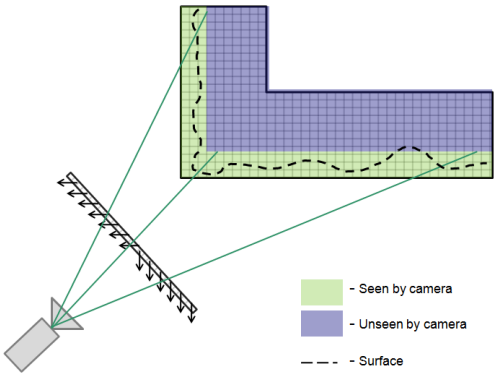


Figure 4: Normal projection from the image plane to the visible band *(green)*. Note, that the surface ∂M is unknown, but we estimate, that it lies within the visible band.

At this point, the computation of the vector field $\vec{V}$ is simple. From each camera, we project the observed normal fields to the visible band of the voxels, as demonstrated in Figure 4. This is efficiently computed using the back-projection of voxels to the image planes. Note, that in case of silhouette carving [17], each voxel is back-projected to the image plane of each camera. Based on a lookup to the silhouette map, a voxel is either kept or rejected, depending on silhouette consistency. In the normal projection case, for each camera, we project all voxels that are in the visible band to the image plane and make a lookup to the color-coded normal map, which is visualized in Figure 5. The obtained normal is then assigned to the voxel being projected. The value at each voxel is divided by the total number of cameras the voxel was seen from. As a consequence of this step, we obtain the vector field $\vec{V}$, which is discretized on a regular grid. Alternatively, this projection process can be seen as normal voting. We assign votes to voxels near the visual hull for being real

Figure 5: A color-coded normal field observed from the i-th camera and the corresponding silhouette.

surface voxels. The next step is the optimization of the flux through the surface represented by the vector field $\vec{V}$ using a Graph-Cut technique, which is described in the next section.

4.3 Energy Optimization using Graph-Cuts

Graph-Cuts have been successfully applied to various computer vision problems such as image restoration [3], stereo vision [3] and segmentation [14]. In [15], it is shown, that it is possible to globally optimize a wide class of geometrically motivated hypersurface functionals with Graph-Cuts. Specifically, it is shown that that any functional that consists of a combination of area/length and flux of the given vector field can be optimized by Graph-Cuts, which is just what we need in our case. Motivated by the successful application of Graph-Cuts to the problem of reconstructing a surface from oriented point clouds [18], we decided to optimize our energy functional (6), also consisting of area and flux term, via Graph-Cuts.

As pointed out in [18], maximizing the flux term of the energy functional (8) is equivalent to optimizing the divergence of the vector field $\vec{V}$ in the interior (Gauss-Ostrogradsky a.k.a. Divergence theorem). Thus, the final energy we are optimizing is:

$$E(\partial M) = \lambda_1 \int_{\partial M} \mathrm{dA} - \lambda_2 \int_M \mathrm{div}(\vec{V}(x))\mathrm{dV}. \qquad (10)$$

This energy can be simply converted into a graph. Adjacent nodes, which are corresponding to voxels in a voxel grid, are connected by *n-links*, and additionally, the nodes are connected to terminals s and t, based on the divergence of $\vec{V}$ (see Figure 6). In the constructed graph, *t-links* are weighted proportional to the absolute value of the divergence at each voxel. Furthermore, voxels with positive volumetric potentials are connected to the source node and voxels with negative volumetric potential are connected to the terminal node. To minimize metrification artifacts, the voxel nodes are linked not only to directly adjacent nodes, but to larger neighbourhoods. The size of the actual neighbourhood can be considered as a parameter. In most of our experiments, we considered neighbourhoods of 26 nodes. The weights of the *n-links* are proportional to the weight λ_1. For detailed information on the weight computation, we refer to [1]. The solution can then be obtained by computing an *s/t-cut* on the graph. In a sense, we are solving

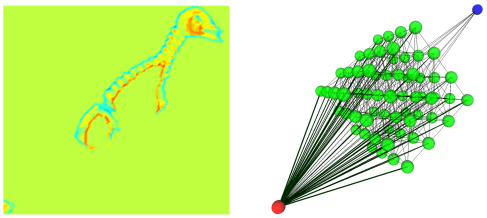


Figure 6: Slice of the computed divergence on the voxel grid from *(left)*. Simple graph constructed on a voxel grid *(right)*. Green nodes correspond to the voxels, red *(leftmost)* node and blue *(rightmost)* node represent the terminals.

a segmentation problem: we are segmenting voxels that belong to the model M from the background.

There exists a variety of efficient algorithms for solving the Min-Cut/Max-Flow problem, starting with Ford-Fulkerson [7] and "push-relabel" [10]. In [2], an exhaustive overview of Min-Cut/Max-Flow algorithms is given, focusing on computer vision problems. The authors of [18] observed, that due to high-resolution grid demands for surface reconstruction, the use of existing Min-Cut/Max-Flow algorithms on a regular grid is practically infeasible and proposed a novel TouchExpand algorithm. Their algorithm computes a global cut while it keeps in memory only local graphs. Because of our regular-grid based discretization, the TouchExpand is method of choice for our surface reconstruction purpose, although it would be an even better idea to use an adaptive grid and thus reduce size of the graph drastically. This way, use of algorithms discussed in [2] would be feasible.

5 Results

To validate our approach, we conducted several experiments. As we do not address the estimation of normal fields but instead assume having such information, we evaluate our approach only on synthetic data, generated with our testing environment, described in Section 5.1. In order to test the robustness of our algorithm, we also added noise to our data sets. We performed tests on the Utah Teapot and Cyberware Dinosaur model (Figure 7).

We implemented our algorithm in Matlab and partially, in C++, and executed it on a PC with a Core2Duo 6600 CPU (2.4GHz) processor and 4GB RAM. While the optimization process takes less than one minute (30 seconds on average), the naive visibility computation can take up to several hours, depending on the number of cameras. In case of 16 cameras, it takes about 1.5 hours.

Because of our Matlab implementation, we had to limit our grid size. The actual resolution depends on the dimensions of the model.

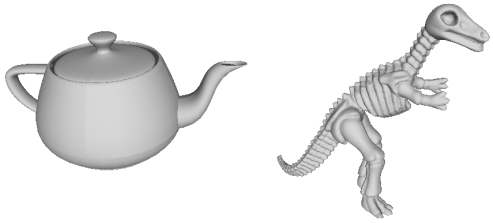


Figure 7: The ground truth: Utah Teapot model *(left)* and Cyberware Dinosaur model *(right)*.

5.1 Testing Environment

We generated our synthetic data sets with a testing environment we developed for the evaluation of our algorithm. The testing application allows the user to place a 3D model into a scene and virtual cameras around it. From these virtual cameras, ground-truth normal images are created by colouring pixels according to their true normals using a simple pixel shader program. In addition, the testing environment is also capable of simulating noisy measurements by adding Gaussian or salt&pepper noise to the rendered normal images.

The generation of synthetic normal fields is not the only purpose of our testing application. We are also planning to make use of it for testing out different techniques for normal estimation of different materials (highly-specular, glass, etc.). Different surface reflectance behaviours can be simply simulated by selecting a different shader program.

5.2 Results on Synthetic Data

In our initial experiments, we used a virtual setup where the object is surrounded by several cameras, as shown on Figure 1. The results using different numbers of cameras and depth parameters ε are visualized in Figures 8, 10, 9 and 11. We observe, that our algorithm is able to produce reasonable reconstructions even when using only a few cameras. Using normal information, also concave regions can be recovered as shown in the Dinosaur example on Figures 8, 9 and 10.

However, the algorithm at this stage is somehow sensitive to the choice of the parameter ε. For thinner bands ($\varepsilon = 3$), it may happen that normals are not projected to the voxels where the surface ∂M actually is located. The effect is visible at the eye of the Cyberware Dinosaur's head in Figure 10. Although the concave region around the eye is recovered to some degree, the algorithm clearly does not reach the desired surface ∂M in this region. In case of choosing deeper bands ($\varepsilon = 6$), the concave region around eye is reconstructed well. For visibility bands of arbitrary depth, reconstruction artifacts (marked with the blue circle) may occur. The reason for this is that in some regions, normal votes from different sides may interfere with each other. How to solve problems related to visibility-band depth parameter will be discussed in Section 6.

Just as in the algorithm presented in [5], our algorithm is also dependent on the initial surface. How to resolve that issue, is also addressed in Section 6. In fact, we believe that with very simple improvements, we will be able to completely skip the silhouette carving step.

In case of Utah Teapot dataset we observe, that the Graph-Cut was able to deal easily with the large uncarved area in the bottom of the Teapot. That region did not receive any normal votes, so the Graph-Cut simply produced the minimal surface at the bottom of the cup. Overall we obtain a nice reconstruction of the Teacup, although we can observe discretization artifacts due to the limited voxel grid size.

In a real-world situation, due to the acquisition process, normal estimates will always contain noise, this is why we corrupted our perfect normal maps with Gaussian and salt&pepper noise. It should also be noted, that normal estimates may be erroneous and that normal fields will not neccessarily match on the real input data. There are also other possible sources of errors that we do not consider here, for example, systematic error due to imprecise camera calibration, errors caused by lighting conditions, outliers, etc. In Figure 12 we can observe, that in presence of light salt&pepper noise and Gaussian noise with standard deviation $\sigma = 0.05$, reconstruction results are not significantly affected. In case of strong Gaussian noise ($\sigma = 0.2$), we observe reconstruction artifacts, but still the algorithm is able to recover the rough shape. From these tests we conclude, that our algorithm is reasonably robust to the noise.

6 Conclusions and Future Work

In this paper, we propose an approach to solve the multiview normal integration problem via Graph-Cuts. For the discretized version of the utilized energy functional under the given approximation of the visibility, our approach produces a globally optimal solution. Furthermore, the energy optimization with Graph-Cuts using the Touch-Expand algorithm is very efficient. The bottle-neck of our approach is the visibility computation, which can be easily optimized and remains our future work. We show, that our algorithm performs reasonably well even on challenging models like the Cyberware's Dinosaur and is very robust to the noise.

One drawback of our approach is its sensitivity to the band parameter and dependence on the visual hull. To resolve this issue, we are planning to implement an iterative approach. In that case, the normals will be projected to the model reconstruction from the last iteration, M_{k-1}, using a very thin visibility band. At each iteration, the visibility will be computed and the Graph-Cut will be applied again to iteratively refine the reconstruction. Such an iterative approach also allows to skip the silhouette carving process. Further improvements should also consider a higher

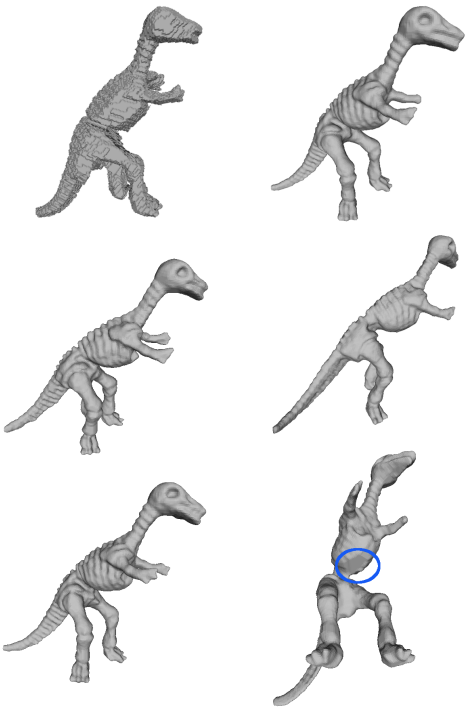


Figure 8: Reconstructions of the Cyberware Dinosaur datasets with band parameter $\varepsilon = 6$. First column: visual hull, reconstruction using 6 cameras and reconstruction using 16 cameras. Second column: reconstructions using 16 cameras. The blue circle marks area that was not seen by the cameras.

grid resolution in order to achieve more accurate reconstruction results.

We also observe, that it is more likely that the surface ∂M coincides with voxels, where projected normals i.e. normal votes match. For that reason, we are planning to incorporate an additional energy functional term, that penalizes differences between the normal votes and the mean normal at each voxel.

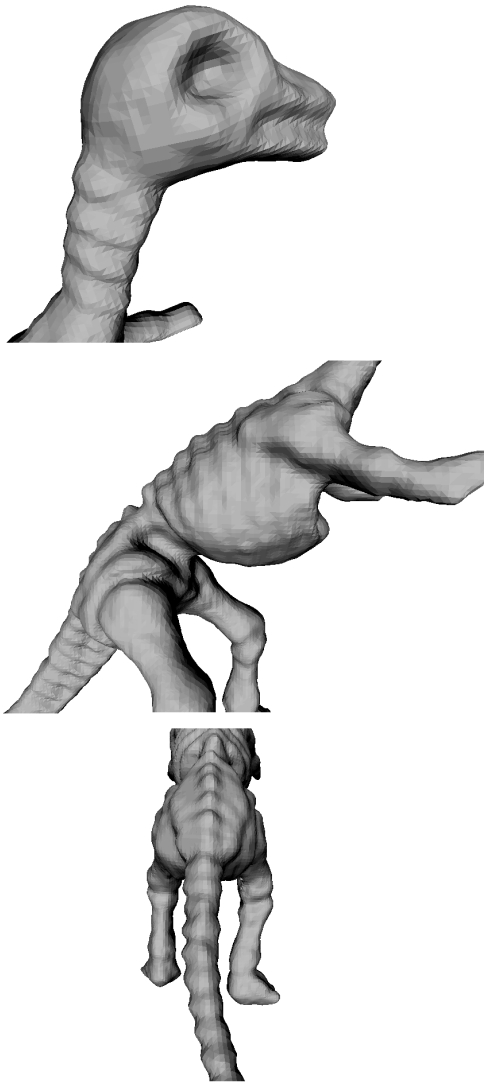


Figure 9: Details of the Cyberware Dinosaur's reconstruction using 16 cameras and $\varepsilon = 6$.

References

[1] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. ICCV '03, 2003.

[2] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE TPAMI*, 26:1124–1137, September 2004.

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE TPAMI*, 23:1222–1239, November 2001.

[4] F. Calakli and G. Taubin. Ssd: Smooth signed distance surface reconstruction. *Comput. Graph. Forum*, 30(7):1993–2002, 2011.

[5] J. Y. Chang, K. M. Lee, and S. U. Lee. Multiview normal field integration using level set methods. In *CVPR'07*, 2007.

[6] T. Chen, M. Goesele, and H. Seidel. Mesostructure from specularity. In *CVPR (2)*, pages 1825–1832, 2006.

[7] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[8] R. T. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE TPAMI*, 10:439–451, July 1988.

[9] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE TPAMI*, 23:643–660, June 2001.

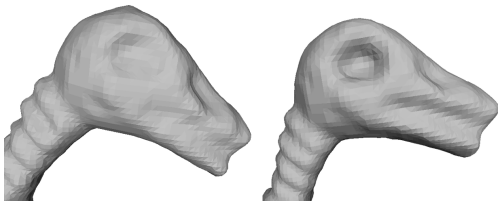


Figure 10: Cyberware Dinosaur's head reconstructed using 16 cameras with $\varepsilon = 3$ *(left)* and $\varepsilon = 6$ *(right)*

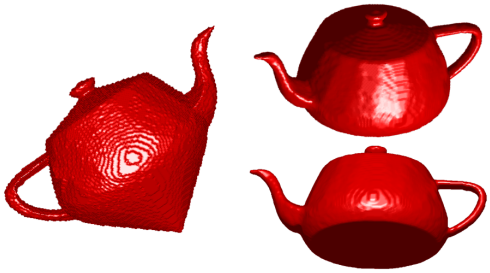


Figure 11: Reconstructions of the Utah Teapot: visual hull *(left)*, result after Graph-Cut *(right)*.

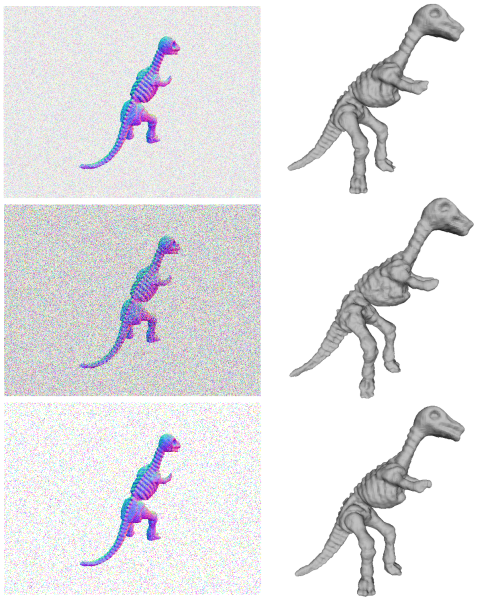


Figure 12: Reconstructions of noisy Cyberware Dinosaur data sets. In first two reconstructions, Gaussian noise with standard deviations $\sigma = 0.05$ and $\sigma = 0.2$ was added. Third reconstruction shows result for salt&pepper noise.

[10] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. STOC '86, pages 136–146, 1986.

[11] C. Hernández and G. Vogiatzis. Shape from photographs: A multi-view stereo pipeline. In Roberto Cipolla, Sebastiano Battiato, and Giovanni Maria Farinella, editors, *Computer Vision: Detection, Recognition and Reconstruction*, volume 285 of *Studies in Computational Intelligence*, pages 281–311. Springer, 2010.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. SIGGRAPH '92, pages 71–78, 1992.

[13] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics symposium on Geometry processing*, SGP '06, pages 61–70, 2006.

[14] J. Kim, J. W. Fisher, III, A. Tsai, C. Wible, A. S. Willsky, and W. M. Wells, III. Incorporating spatial priors into an information theoretic approach for fmri data analysis. MICCAI '00, pages 62–71, 2000.

[15] V. Kolmogorov and Y. Boykov. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. ICCV '05, pages 564–571, 2005.

[16] P. Kovesi. Shapelets correlated with surface normals produce surfaces. ICCV '05, pages 994–1001, Washington, DC, USA, 2005. IEEE Computer Society.

[17] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE TPAMI*, 16:150–162, February 1994.

[18] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *CVPR, 6*, 2007.

[19] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987.

[20] J. Manson, G. Petrova, and S. Schaefer. Streaming surface reconstruction using wavelets. *Computer Graphics Forum (Proceedings of the Symposium on Geometry Processing)*, 27(5):1411–1420, 2008.

[21] D. Reddy, A. K. Agrawal, and R. Chellappa. Enforcing integrability by error correction using l1-minimization. In *CVPR*, pages 2350–2357. IEEE, 2009.

[22] T. Simchony, R. Chellappa, and M. Shao. Direct analytical methods for solving poisson equations in computer vision problems. *IEEE TPAMI*, 12:435–446, May 1990.

[23] R. Zhang, P. Tsai, J. E. Cryer, and M. Shah. Shape from shading: A survey. *IEEE TPAMI*, 21:690–706, August 1999.

Priority-Based Task Management in a GPGPU Megakernel

Bernhard Kerbl*
Supervised by: Markus Steinberger†

Institute for Computer Graphics and Vision
Graz University of Technology
Graz / Austria

Abstract

In this paper, we examine the challenges of implementing priority-based task management with respect to user-defined preferential attributes on a graphics processing unit (GPU). Previous approaches usually rely on constant synchronization with the CPU to determine the proper chronological sequence for execution. We transfer the responsibility for evaluating and arranging planned tasks to the GPU where they are continuously processed in a persistent kernel. By implementing a dynamic work queue with segments of variable size, we introduce possibilities for gradually improving the overall order and identify necessary meta data required to avoid write-read conflicts in a massively parallel environment. We employ an autonomous controller module to allow queue management to run concurrently with task execution. We revise Batcher's bitonic merge sort and show its eligibility for sorting partitioned work queues. The performance of the algorithm for tasks with constant execution time is evaluated with respect to multiple setups and priority types. An implementation of a Monte Carlo Ray-Tracing engine is presented in which we use appropriate priority functions to direct the available resources of the GPU to areas of interest. The results show increased precision for the desired features at early rendering stages, demonstrating the selective preference established by our management system.

Keywords: GPGPU, megakernel, GPU sorting, priority-based task management, dynamic work queue, Monte Carlo Ray-Tracing

1 Introduction

Parallel computing has become a valuable tool for handling time-consuming procedures that contain a high number of homogeneous, independent operations. Since the rise of the Unified Shader Model which features explicit programmability of GPUs, exploiting data level parallelism on a customary personal computer has become increasingly straightforward. The advent of NVIDIA's GeForce 8 Series introduced the first release of the CUDA architecture and associated compilers for industry standard programming languages. With respect to certain restrictions, the architecture enables programmers to run general purpose computations on compatible devices in parallel. Newer models of GPUs supporting these features are therefore often referred to as general purpose graphics processing units (GPGPU). Several hundred Stream Processors (SP), also referred to as thread blocks, which are grouped in clusters called Streaming Multiprocessors (SM) can be instructed to execute commands at the same time – the programmer simply specifies the number of necessary threads when launching a GPGPU kernel. The combined capacities of these SPs have produced a significant gap in raw speed between high-end GPUs and CPUs [12]. However, using the available resources to their full potential is not an easy task. Memory latencies, insufficient parallelization or unfavorable occupancy at runtime may cause implementations to fall short of their ideal behavior.

One particular cause of poor efficiency in CUDA kernels is unbalanced work load distribution, which can have a limiting effect on performance, especially when considering problems that exhibit irregular behavior, e. g. adaptive ray-tracing techniques [2]. Instead of relying on the built-in CUDA work distribution units, custom task management strategies can be implemented to address these issues [4]. One specific example is given in the OptiX Ray-Tracing Engine and its dynamically load-balanced GPU execution model [14].

In this context, the term *megakernel* refers to a solution where improved work load distribution is achieved using a persistent kernel in order to benefit from uninterrupted computational activity. One or more queues are commonly used to store tasks that are constantly being fetched and executed until the queues are empty. For static work queues, tasks can only be added in between megakernel launches, while dynamic implementations also allow for insertion of new tasks at runtime [4].

When using a megakernel for processing diverse problems and the implied task level parallelism, assigning meaningful priorities to each task can have a positive effect. Especially procedures that involve rendering may experience a considerable boost in usability through proper task classification. Considering applications with guaranteed frame rates, the perceived quality of each frame can

*kerbl@student.tugraz.at
†steinberger@icg.tugraz.at

be improved by focusing on areas that exhibit prominent features [9]. High quality initial views of rendered scenes can be generated by directing the resources of the GPU based on priorities. In an interactive environment with a modifiable view model, early perception of the visible dataset enables efficient adjustment of the extrinsic parameters in order to obtain a desired setup. Similarly, prioritizing user interactions achieves faster response to input commands and postpones time-consuming rendering procedures that are otherwise wasted on a setup that is inadequate.

Previous approaches based on priorities commonly exploit the sophisticated sorting methods on the CPU and establish means for communicating the favored order of execution to the GPU [10, 6]. Moving the responsibility of organizing the runtime agenda to the GPU expectably eliminates the otherwise significant overhead of inter-component communication. Therefore, we aim to provide a functional autonomic task management system for dynamic work queues on the GPU, in order to enable adaptive behavior for CUDA applications without interrupting the execution of tasks.

2 Previous Work

While analyzing the influences of possible limiting factors on ray-tracing kernels, Aila and Laine experimentally bypassed the default CUDA work distribution units, suggesting that this approach might show an improvement for tasks with varying durations [2]. In their implementation, they use a work queue from which tasks are constantly fetched and executed in individual thread blocks, which proved to be very effective. A detailed analysis of possible techniques for using work queues in a GPGPU kernel was authored by Cederman and Tsigas, addressing sophisticated load balancing schemes which are made possible through the atomic hardware primitives supported by the CUDA architecture [4].

In [14], Parker et al. successfully employed a self-provided strategy for balancing work load with the goal of improving efficiency in a ray-tracing engine using a persistent megakernel. Furthermore, they use static prioritization to achieve fine-grained internal scheduling for tasks to avoid penalties resulting from state divergence.

A recent example for a priority-based solution that achieves GPU task management in real-time is TimeGraph [10]. The routine relies on interrupts and substantial communication between the GPU and the CPU which acts as an executive supervisor. A similar system was implemented by Chen et al., in which queues are shared and accessed regularly by the CPU and GPU [6]. In recent developments, Kainz et al. employed prioritization in a rendering application by sorting work queues between kernel launches to achieve improved richness of detail for predetermined frame rates [9].

Efficient sorting algorithms for parallel systems, especially targeting GPGPUs, have become a popular research topic. Following the implementations of Batcher's bitonic merge sort which was developed for parallel sorting networks [3], other algorithms designed for use on massive datasets were adopted for the GPU as well [5, 7, 13, 15].

3 GPU Megakernel System

3.1 Available Resources and Challenges

We target dynamic GPGPU task management by introducing a global, self-organizing work queue in a megakernel system which allows for arbitrary tasks to be added at runtime. Apart from the functions to be invoked upon execution of a task, each queue entry provides additional attributes that are considered during different stages of the management process. Sorting the queue is a time-critical problem, since the megakernel system constantly removes the frontmost entries and processes the associated behavior in the available SPs to achieve proper workload distribution.

Due to the intended autonomy of our management system, we cannot rely on the CPU to rearrange queue entries based on their priorities. Instead, one thread block is reserved and used as a controlling unit which is responsible for sorting the queue at runtime. The remaining thread blocks are henceforth referred to as *working module* to clarify their intended purpose. Since both modules constantly process the shared contents of the queue, leaving them unprotected would cause interference and lead to severe runtime errors. Therefore, we require secure methods for classifying queue entries and deciding whether they are safe to be processed by either module.

3.2 Work Queue Segmentation

Since the contents of a dynamic work queue are potentially volatile, we partition the queue and thereby enable faster detection of segments that are unlikely to change in the near future. Consequently, the controller can quickly select segments that are not yet in use and perform sorting while the working module constantly removes entries from segments in the front of the queue. Treating segments as instances of classes gives us the advantage of storing additional information about the contained queue entries as attributes, such as a reference to the task with the shortest execution time or current availability.

For using segments to restructure the work queue, we identify two additional constraints in order to avoid access conflicts. First, only full segments qualify as appropriate candidates for sorting, which leads to entries at the back being ignored if they are located in a partially populated segment. Second, since we must not tamper with the entries of segments whose contents are currently being executed, we cannot sort the tasks at the very front, which leads to inevitable disarray for the leading segments.

Considering these limitations and their effects on kernel execution, we provide settings to adapt the behavior of the controller as required. The user may select a specific segment size for a particular purpose: while a smaller size generates a finer granularity and reduces the number of neglected entries both in the front and in the back of the queue, it requires high organizational overhead and increases the time needed for establishing a decent order. A bigger segment size may improve the sorting performance for a higher number of queue entries, but at the same time larger chunks of the queue will be ignored due to incomplete segments.

3.3 Mutual Exclusion

Even though we established that unused segments can be quickly identified by the controller using only work queue partitioning, we need to consider overlapping accesses since sorting algorithms are time-consuming as well and may take even longer than task execution. Dealing with these situations requires an unambiguous system-wide regulation for deciding which module is currently allowed access to a segment. We decided to introduce a thread-safe policy for checking and setting the current availability state of a segment, combining two common approaches in a multipurpose attribute variable.

The CUDA instruction set provides means to change the contents of the attribute variable atomically. Each SP in the system tests the state of a particular segment before accessing its contents. For the working module, the state variable behaves like a semaphore, allowing a specified number of accesses before resetting it to the initial zero value. If the controller requires the contents of an available segment, it exchanges the current state with a negative integer. Threads in the working module that are trying to acquire the contents of segments that are being sorted will perform a busy wait until the controller signals completion of the sorting algorithm by assigning a positive value to the variable.

4 Sorting the Queue

4.1 Bitonic Merge Sort

Our controller utilizes an adaptation of the bitonic merge sort to rearrange the contents of the work queue. The algorithm was devised by Ken Batcher for parallel sorting networks as an alternative to the odd-even merge sort and operates by constructing bitonic sequences of increasing lengths and merging them to generate sorted output [3]. A simple setup demonstrating the procedure for applying the algorithm in parallel is illustrated in Figure 1. For an array of length N and T concurrent threads, the algorithm requires $\mathcal{O}(\frac{N}{T} \cdot log^2 N)$ parallel comparison operations. Popular fields of application include collision detection and visibility calculation in particle systems [11]. The most

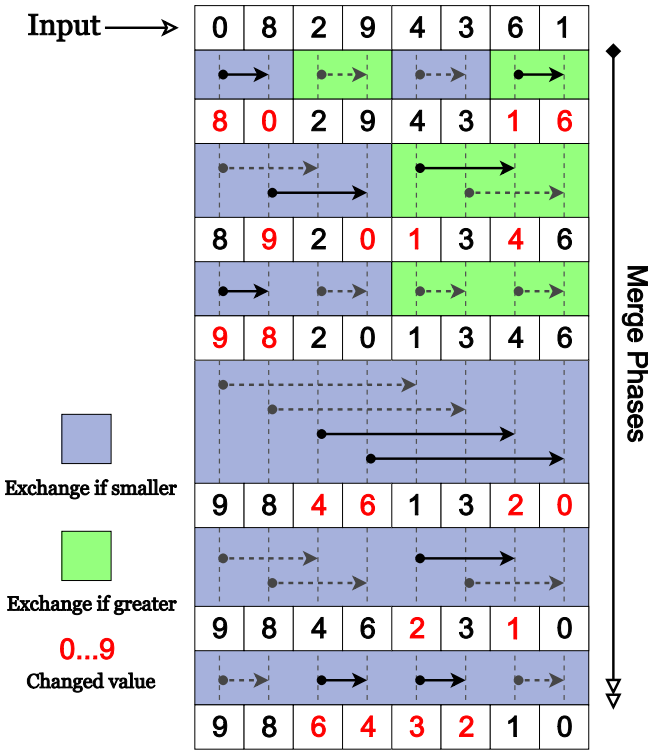


Figure 1: One possible implementation of the bitonic merge sort using one thread per element index. The colored rectangles indicate different comparison operators being used for evaluating whether two values should be exchanged. By forming bitonic sequences in each step, the merge phases are applied consecutively until the input is sorted in descending order.

efficient GPU implementations of the bitonic merge sort were able to outperform the sophisticated std::sort methods on contemporary CPUs [15]. Recently developed algorithms for sorting on the GPU are more commonly based on radix or bucket sort, although some frameworks implement hybrid variants in order to exploit the characteristics of bitonic sequences [13, 7, 16].

4.2 Benefits

We aim to provide the user with the possibility of choosing custom priority values. The bitonic merge sort is comparison based and is therefore applicable for any data type that supports the logical operators $<$ and $>$.

For smaller data sets, the bitonic merge sort is one of the fastest algorithms if the underlying architecture is optimally exploited [8]. Ideally, the thread block size equals the number of elements to be sorted. In such a case, $\frac{N}{T} = 1.0$ and the algorithm requires exactly $log^2 N$ parallel steps to execute.

Concatenating two arrays sorted in distinct order yields a bitonic sequence by definition, which corresponds to the input in the final top-level merge phase of the bitonic

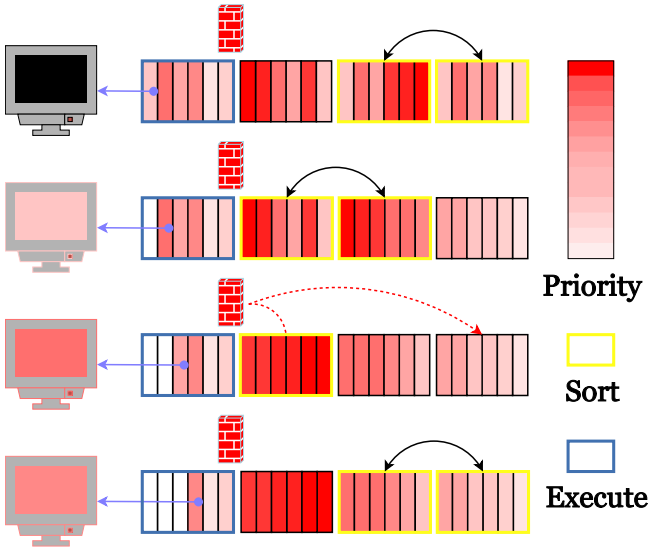


Figure 2: A pair-wise sorting algorithm applied to the segments in a queue. Neighboring segments are combined to achieve gradual restructuring of the contents, starting from the back and advancing towards the front. If a segment is encountered that is currently unavailable, the algorithm restarts from the back.

merge sort. Based on these properties, we can reduce the number of necessary comparison operations significantly when sorting two preprocessed arrays by inverting one array and appending it to the other, thereby creating a bitonic sequence. For fusing two sorted arrays of length $\frac{N}{2}$, the worst-case complexity can thus be expressed as $\mathcal{O}(\frac{N}{T} \cdot log\, N)$.

4.3 Using Bitonic Merge Sort on Segments

We use an optimized bitonic merge sort to rearrange segments of the work queue sequentially in successive passes. For each pass, we need to acquire available segments and apply the bitonic algorithm to their combined data set. The key-index pairs are retrieved from the work queue using the segment indices as offsets for addressing the associated tasks. We identified two basic qualities that the controller should exhibit when sorting segments:

- Constant refinement: the accuracy of the resulting sequence should increase with the number of passes
- Effectiveness: high-priority work packages should advance towards the front as soon as possible

The basic idea to improve the overall order is to compare each segment with its predecessor iteratively. For a work queue with N segments of size S, we need $N-1$ passes to move the S most important tasks to the leading segment. This approach, though basic, continually improves the order in the queue. If we reach a segment where no predecessor can be acquired, we reinitiate the procedure starting from the back, which is illustrated in Figure 2. The resulting accuracy of the sort is proportional to the number of passes performed. Obtaining a fully sorted list would require a total of $\frac{N^2+N}{2}$ passes, but approximate sorting with an emphasis on prioritizing important tasks over regular ones is sufficient to induce adaptive behavior. Also, as mentioned in Section 4.2, partially sorted input can be processed much faster. This property translates well to the segment-based approach: by monitoring a boolean member variable that is true for segments that are revisited, we can decide whether it is sufficient to apply a top-level bitonic merge. We consider three different combinations of segments and provide an optimized sorting method for each of the following pairings:

- **Unsorted – Unsorted**
- **Unsorted – Sorted** or **Sorted – Unsorted**
- **Sorted – Sorted**

5 Time Management

5.1 Motivation

The system described thus far is capable of managing queued tasks based on their priorities without assistance by external components. However, due to the necessary synchronization of controller and working module by mutual exclusion, a significant delay is added to the total megakernel execution time whenever a SP in the working module transitions into a state of busy waiting, caused by segments being unavailable as they are currently being sorted. We target this issue by implementing a management strategy using time-based regulations for avoiding collisions of the working module and the controller when accessing the work queue. The chosen approach requires collection and maintenance of related meta data for task duration and sorting performance.

5.2 Avoiding Collisions

Usually, the primary objective of a megakernel is to exploit the resources of the GPU. Hence, we do not put any additional constraints on accesses made by the working module. Instead, the controller performs advanced sanity checks before locking two segments for sorting. Following the algorithm described in Section 4.3, we select two qualified segments. The time needed for performing the bitonic merge sort on the selected segments is stored in $time_{sorting}$. We then proceed to probe the anterior sections of the queue: segments in between the chosen candidates for sorting and those that are currently being executed are regarded as time buffers. The required amount of time for the working module to process a buffer segment is estimated by the execution time of the shortest contained

task. The respective variables are read for each segment and accumulated from back to front in a second variable $time_{buffers}$. Once the condition $time_{buffers} \geq time_{sorting}$ is fulfilled, we abort the traversal stage and initiate the bitonic merge sort procedure. If the available time is not sufficient for sorting, the procedure is reinitiated at the back of the queue to prevent possible collisions.

5.3 Collecting Meta Data

5.3.1 Task Execution

For each task, the working module records the time needed for execution and compares it to previous results by default. It is necessary to monitor these values constantly for automatic runtime detection since the execution time of a task may fluctuate considerably and cannot be predicted without error. Whenever a new task is added to the work queue, the corresponding segment updates its estimated buffering effect based on these measurements. In order to minimize the probability of collisions, we choose the pessimistic approach and exclusively store the shortest duration for each task measured so far.

5.3.2 Sorting Methods

For each of the three sorting modes, we measure and update the number of clock cycles needed by the respective method whenever it is invoked. The $time_{sorting}$ variable can thus be estimated more accurately by the controller based on the orderliness of tasks in the segments that are selected for sorting. As opposed to task execution, measurements are discarded if they are lower than previous results, although sorting performance is less likely to change over time. This approach further reduces the likelihood of collisions in the work queue, since always the longest duration is assumed for each sorting method.

5.4 Custom Timing Settings

As an alternative to the default pessimistic behavior, we provide settings for the advanced user to customize the internal time management. A more flexible strategy can be achieved by defining a global multiplier for the time buffers. If a task is known to have a reliable average duration or requires a fixed number of clock cycles defined by its inherent complexity, the automatic runtime detection may be disabled selectively. Instead, a static value can be provided to indicate how many clock cycles should be assumed for execution.

For projects where priority sorting is of utmost importance, meta data acquisition may be disabled for all tasks. The user can then define a constant global variable that is substituted for each buffer segment.

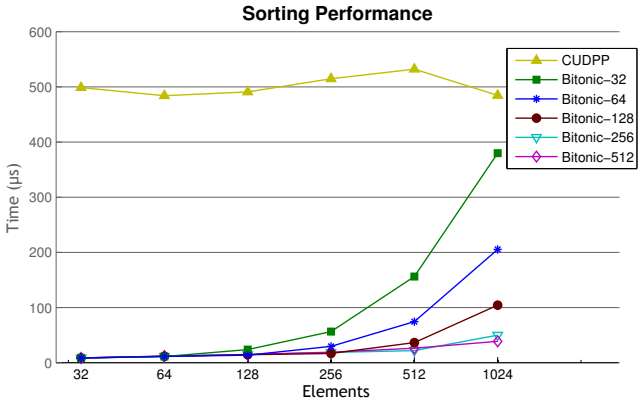


Figure 3: Comparison of performance for the tested sorters with the standard CUDPP radix sort for unsigned integers. Our implementation of the bitonic merge sort yields almost constant results for setups where the number of elements does not exceed the number of threads used.

6 Results

6.1 Bitonic Merge Sort Performance

The performance of the bitonic merge sort in our system is based on two factors, namely the segment size and the number of active threads. The segment size can be modified in order to achieve a desired granularity for the sort. Furthermore, the selected block size for a kernel also defines the dimension of the controller. The algorithm was therefore evaluated using a representative selection of settings. In order to compare its potential efficiency to existing sophisticated routines, the optimized bitonic merge sort was executed using a single thread block for sorting a limited number of keys outside of the megakernel system. This procedure is equivalent to a sorting pass of the controller module and thus models the expected behavior for sorting two segments. We chose thread block dimensions and array lengths as powers of 2. Thread block dimensions range from common CUDA warp size of 32 threads to the largest possible block size of 512 threads. Array lengths start at 32 and end at 1024 elements, which equates to double the maximum segment size in our megakernel system. The tests were conducted using a GeForce 560 Ti. A visualization of the parametrized setups and the resulting runtimes for sorting unsigned integers can be found in Figure 3. We compared our results with the recorded times from the CUDPP test suite [1]. Even for our most unfavorable setting with 32 threads sorting an array of 1024 values, the optimized bitonic merge sort beat the CUDPP radix sort by little over 100 μs ($\sim$ 20%). For more balanced setups, our implementation was up to 56 times faster in comparison. We would like to point out that the CUDPP project targets larger data sets and is indeed very potent for lengthy input [16]. Considering the non-linear growth rates, the emergent trend suggests that with increasing array sizes the bitonic sorter will eventually be bested. How-

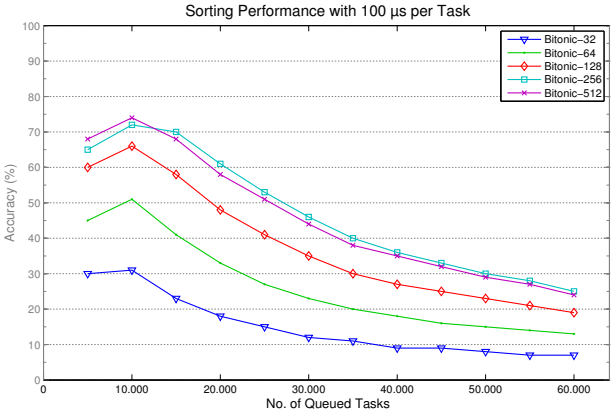


Figure 4: Algorithm performance with a fixed task duration of 100 μs. Starting at 10,000 tasks, we observe a steady decline in accuracy. This suggests that although the negative influence of unreachable segments is constantly reduced, the performance eventually drops due to insufficient time buffers.

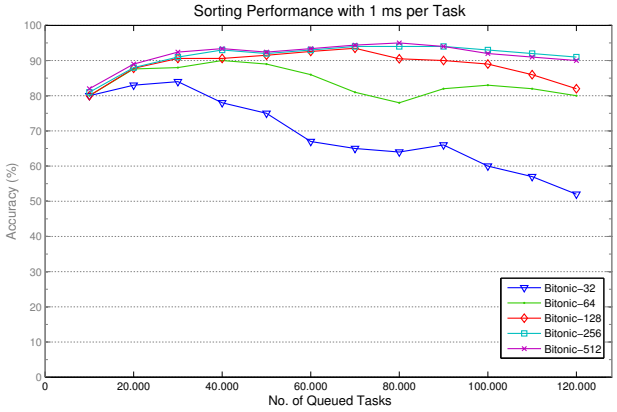


Figure 5: Evaluation of the algorithm performance for complex tasks with runtimes exceeding 1 ms. For smaller thread block sizes, we observe an early decline due to the increased effort for sorting a high number of entries with fewer threads. The highest recorded score of 95% is obtained using 512 threads.

ever, given that we intend to frequently sort pairs of segments containing 512 tasks or less using only one thread block, it appears to be a very suitable solution.

6.2 Sorting the Queue at Runtime

We evaluated the performance of the presented strategy for sorting the queue segment-wise in our megakernel system. Since the duration of a megakernel is implicitly determined by the tasks it executes, we assessed the accuracy of our management system for a given number of tasks instead. We used blocking tasks to occupy thread blocks for a specified number of clock cycles. The segment size for the work queue was set to 256 tasks in order to balance granularity and sorting efficiency. Since both modules in the megakernel start simultaneously, the leading segments are immediately locked by the working module and can never be processed by the controller, which makes it impossible to achieve a fully sorted queue at runtime. Regarding these constraints, we estimated the actual performance by rating each executed task based on its predecessors. We enabled automatic runtime detection and stored the priorities of executed tasks chronologically in a list of entries L which was subsequently evaluated. Ideally, we anticipate a descending sequence of values where an entry at index i (starting from 0) has i previous entries representing tasks with higher priorities. Hence, the score for each tested setup with the specified number of tasks n is defined as follows:

$$S(n) = \frac{\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} f(i,j)}{n-1}, \quad f(a,b) = \begin{cases} \frac{1}{a}, & \text{if } L(b) \geq L(a) \\ 0, & \text{else} \end{cases}$$

Assuming worst case conditions for our test setup, the queue was initiated with task priorities in ascending order, which would yield a total score of 0. The results for a given number of entries where each associated task took at least 100 μs are illustrated in Figure 4. The accuracy of the order in which they were executed peaked at 74% when using a block size of 512 threads for 10,000 tasks. Lower values preceding the apex of each function were caused by the statistical influence of the reserved leading segments which were not sorted before execution.

Due to device-related delays between fetching and processing, tasks may not be executed precisely in the same order as they are ranked in the queue. The resulting effect caused slightly lower scores when using 512 threads compared with a block size of 256 threads for a higher number of tasks. For time-consuming procedures, such as those found in elaborate ray-tracing engines, we considered the results after raising the blocking interval to 1 ms (see Figure 5). With the highest possible number of threads, we achieve a maximum score of 95% for 80,000 entries. Based on these results, we can conclude that the accuracy in the order of execution increases with the complexity of planned tasks, since the prolonged execution time can be utilized to issue additional sorting passes.

6.3 Adaptive Monte Carlo Ray-Tracing

We demonstrate the effects of using custom priorities in a progressive Monte Carlo Ray-Tracing engine and two different scenes. The engine evaluates 512 paths per pixel with a resolution of 800x600 and checks intersections with objects iteratively. Pixels are grouped as patches of 4x4 and each patch is assigned to a designated task instead of using one task for each pixel. The reduced number of work queue entries to be sorted improves the efficiency of the task management system. A task that is executed computes two random traversal paths for each pixel in a patch

and evaluates the new priority based on different strategies, two of which are presented in this section. Tasks repeatedly append themselves to the end of the queue until all rays for the associated pixels have been cast. We provide representative snapshots of both scenes after executing ∼60% of all planned tasks. In order to illustrate the associated progress, we use heat maps to indicate the number of rays cast for each pixel patch.

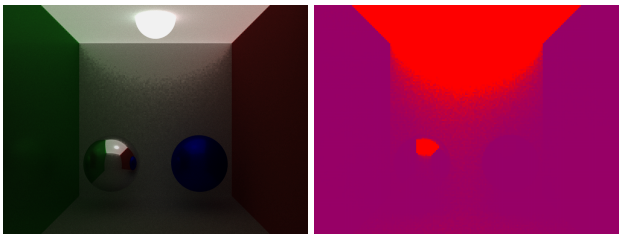


Figure 6: Snapshot of a scene containing objects whose eventual surface colors are influenced by illumination and reflection. Prioritizing patches with high accumulated intensity leads to selective rendering of light sources and white surfaces, which enables a faster perception of details in these areas.

In our first test case, color values returned by rays for each pixel were simply accumulated and the image was then rendered using maximum to white mapping. The examined scene shows the interior of a softly lit room containing reflecting objects and light sources. We used our task management system to focus on high intensity color values. This eventually led to a global preference of brighter areas. Figure 6 shows the scene at an intermediate stage. We can clearly discern the objects that are emitting or reflecting light and observe the increased detail for the left sphere and the ceiling. The priority for each patch was calculated using $priority = \sum_{i=1}^{pixels} color$.

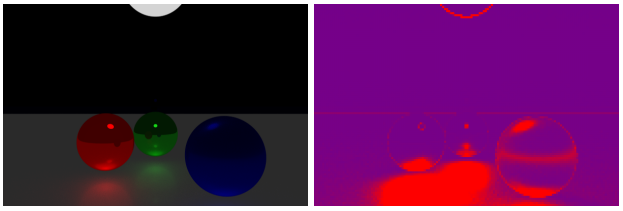


Figure 7: Using difference values as priorities leads to preference of patches with low convergence rates. We observe more rays being cast early on for reflections of spheres on the floor since the surface generates diverse color values depending on the direction of rebounded rays.

For our second example, the output was normalized at each pixel using the heat map data. The selected setup allowed for detection of pixel patches with low convergence rates and prioritization of these areas to achieve enhanced initial views. A large portion of the scene could be neglected at first due to an opaque, unlit wall at the far end of the room (see Figure 7). In Figure 8, we emphasize improved image quality when compared with uniform rendering by magnifying the affected regions of images generated 210 ms after initiation. The applied priority formula can be as expressed as $priority = \sum_{i=1}^{pixels} (color - color_{old})$.

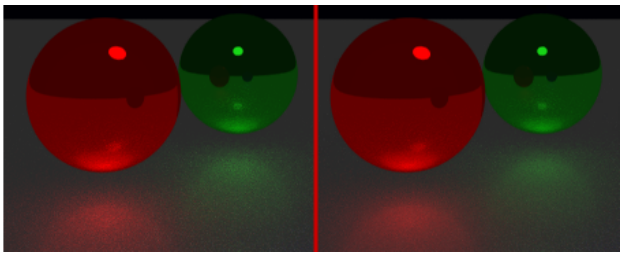


Figure 8: We compare the conventional approach of distributing rays uniformly with the priority-based procedure. The left side shows an early closeup of the scene using the default method and exhibits more noise than the image on the right, which features smooth color reflections as a result of proper task management.

We used the normalized output images from the second test case to assess the mean squared error (MSE) for default and priority-based rendering when compared to the ground truth. Even though evaluating the priority formula required expensive atomic operations, we noticed a clear reduction of the MSE at each point in time (see Figure 9).

7 Conclusion

We presented a priority-based task management system with elaborate timing strategies to enable adaptive behavior for GPGPU programs. We successfully incorporate a controller module in a megakernel system and prevent it from interfering with the continuous execution of queued tasks. We eliminate inter-component communication and the associated overhead by sorting the contents of our dynamic work queue at runtime using a designated thread block. By optimizing the bitonic merge sort algorithm, we establish a basis for iteratively rearranging the segments of the queue. We evaluate the effectiveness of the sorting algorithm by invoking large numbers of tasks and compare performance for tasks with different durations. For more complex tasks, we reach promising scores regarding the order of execution even in unfavorable setups. A Monte Carlo Ray-Tracing engine running in our system shows adaptive behavior and demonstrates how prioritization in a rendering application can speed up the assessment of desired features in a scene. Adaptive rendering offers an extensive field of research for possible priority functions and their impact on image quality. Since the effects of complex formulas do not necessarily outweigh the corresponding overhead, the development of new, efficient priorities requires profound research and sophisticated methodology.

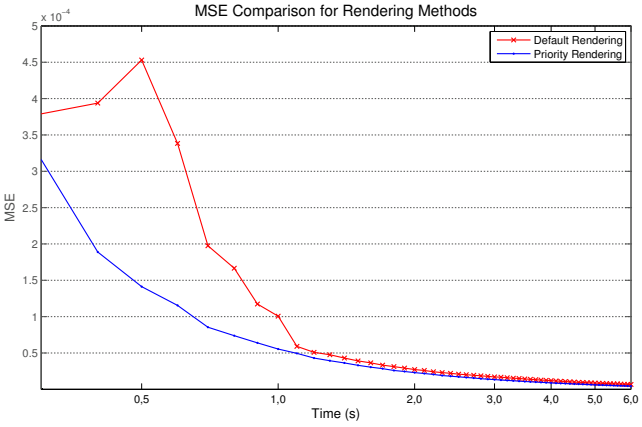


Figure 9: We calculate the mean squared error by subtracting snapshots of the rendered scene generated at regular intervals from the ground truth and evaluating the difference of pixel values. Prioritization of pixel patches that return ambiguous color values leads to improved results for our second test case.

References

[1] Cudpp: Cuda data-parallel primitives library, 2012. `http://gpgpu.org/developer/cudpp`.

[2] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM.

[3] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[4] Daniel Cederman and Philippas Tsigas. On dynamic load balancing on graphics processors. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '08, pages 57–64, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[5] Daniel Cederman and Philippas Tsigas. On sorting and load balancing on gpus. *SIGARCH Comput. Archit. News*, 36:11–18, June 2009.

[6] Long Chen, Oreste Villa, Sriram Krishnamoorthy, and Guang R. Gao. Dynamic load balancing on single- and multi-gpu systems. In *IPDPS*, pages 1–12. IEEE, 2010.

[7] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. Gputerasort: high performance graphics co-processor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 325–336, New York, NY, USA, 2006. ACM.

[8] Mihai F. Ionescu. Optimizing parallel bitonic sort. In *Proceedings of the 11th International Symposium on Parallel Processing*, IPPS '97, pages 303–309, Washington, DC, USA, 1997. IEEE Computer Society.

[9] Bernhard Kainz, Markus Steinberger, Stefan Hauswiesner, Rostislav Khlebnikov, and Dieter Schmalstieg. Stylization-based ray prioritization for guaranteed frame rates. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 43–54, New York, NY, USA, 2011. ACM.

[10] Shinpei Kato, Karthik Lakshmanan, Ragunathan Rajkumar, and Yutaka Ishikawa. Timegraph: Gpu scheduling for real-time multi-tasking environments. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, Berkeley, CA, USA, 2011. USENIX Association.

[11] Peter Kipfer, Mark Segal, and Rüdiger Westermann. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM Press.

[12] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[13] Hubert Nguyen. *GPU Gems 3*. Addison-Wesley Professional, first edition, 2007.

[14] 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.

[15] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[16] Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore gpus. NVIDIA Technical Report NVR-2008-001, NVIDIA Corporation, September 2008.

Cameras & Materials

On Rendering with Complex Camera Models

Bohumír Zámečník*
Supervised by: Alexander Wilkie†

Faculty of Mathematics and Physics
Charles University
Prague / Czech Republic

Abstract

One of the areas of realistic image synthesis is in modeling cameras. The goal is to provide a visual cue with depth of field and to achieve a photographic look with bokeh (out-of-focus highlights), tilt-shift and optical aberrations of real-world lenses. We provide a comparison of existing methods and fundamental approaches for depth-of-field rendering, including the recent methods, such as the image-based ray tracing. We propose a novel representation of ray transfer within complex lenses suitable for optimizing the ray generation. The open problems in this research area are presented along with sketches of possible solutions.

Keywords: Depth of Field, Lens, Photorealistic Rendering

1 Introduction

One of the areas of photo-realistic image synthesis is the image capture via camera models. Both human eyes and photographic cameras naturally depict some parts of the scene sharply while the rest is gradually blurred. It helps the viewers perceive the spatial arrangement of the scene. Simulating this depth of field as well as other effects caused by more or less complex lens systems in a physically plausible way helps in achieving more realism in rendering.

The paper is structured as follows: In chapter 2 we introduce the desired effects and the camera models used in photo-realistic rendering. Chapter 3 summarizes the most important approaches and methods for depth-of-field rendering and makes a comparison. In chapter 4 we propose a novel representation of ray transfer within complex lens systems. And finally the major open problems we found in this area are presented in chapter 5.

*bohumir.zamecnik@gmail.com
†alexander@wilkie.at

2 Camera models

In order to make useful images of a radiance field in the scene we use models conceptually based on real-world cameras. A *camera* consists of a sensor and an aperture or a lens system. The sensor is usually a rectangular grid of pixels which accumulate the incoming radiance to compute the total radiant energy. The aperture or lens system limits and/or transforms the rays of light going to the sensor in order to make an image of a part of the scene.

The most basic and widespread is model of an ideal pinhole with a point-sized aperture. It implements the perspective projection and produces all-sharp images. In practice a finite aperture is needed to pass enough light in. The solution to provide sharp images is in using refractive lenses able to focus rays emanating from a point in the scene to a point on the sensor so that a contribution of multiple light paths can be integrated.

The idealized model of a refractive lens is the thin lens model. It can be either described by a 1st order approximation of the Snell's law of refraction or by a matrix transformation in homogeneous coordinates. Generally the lenses trade off allowing more light paths for being unable to image the whole scene sharply. In the thin lens model just a single plane is sharp, the *focus plane*, the image of the sensor plane via the lens transformation.

Basically the sensor plane is perpendicular to the viewing direction, the optical axis which intersects its center. Focusing can be done by moving the sensor back and forth in the optical axis. The sensor is called *shifted* in case it is moved laterally (within the sensor plane) and *tilted* if it is not oriented perpendicularly to the optical axis. We call this the camera configuration.

In contrast to most consumer cameras the older view cameras and some special or home-made lenses offer tilt-shift configurations. This enables the photographers to focus on an arbitrary plane or change the perspective, which can be useful for artistic purposes.

Points on the focus plane are projected at points on the sensor, while the rest as sections of the cone of light through the aperture with the sensor plane. For non-tilt configurations of the thin lens model the out-of-focus points are imaged as circles (also called *circles of confusion* – CoC) which leads to blur.

The perceived sharpness depends on the spatial resolution of the sensor. For a fixed camera configuration the CoC radius as a function of position in the scene makes up a 3D scalar field. The isosurfaces of this field mark out the boundary of the *depth of field* (DoF). By limiting the maximum blur amount we get a region bounded by the corresponding isosurfaces and objects within such a region can be considered in-focus.

The intensity of images of out-of-focus points quickly decreases with the amount of defocus as the radiant power is spread over a quadratically larger area. For a very bright point light its out-of-focus image is clearly distinguishable and is called *bokeh* in photographic jargon.

The thin lens model is not capable of producing all the characteristic effects caused by physical design of real-world complex lens systems. Due to physical reasons, technological trade-offs and the usage of real-world materials such lenses might not be always capable of perfect focusing. This leads to optical aberrations and geometric distortions and it also has an impact on the imaging quality including the bokeh quality.

In optical engineering as well as in computer graphics the complex lens systems are described by a sequence of analytical surfaces, their mutual position and materials after each surface [17, 28]. Predominantly are used spherical caps for lens surfaces and circles or other shapes for diaphragms. It is the diaphragm shape which affects the bokeh quality the most.

Other lens properties such as coatings and lens effects such as lens flare (caused by internal reflections and diffraction) are out of the scope of this paper. More information can be found in [28, 15].

3 Methods of DoF rendering

In the ideal pinhole model the light goes through a singular center of projection. In contrast, other lens models allow the light paths to pass through a finite area of the aperture stop.Thus all depth-of-field rendering algorithms when solving the rendering equation [16] or its approximation must additionally integrate the light transfer over this area.

There are two main approaches how rendering algorithms solve visibility, i.e. deciding which scene primitives contribute to each pixel and vice versa. They differ in the order of nested loops over scene objects and image pixels [12]. Object-based algorithms compute the illumination for each object for each pixel and image-based ones conversely. Scan-line rasterization is an example of the first approach while ray tracing and its variants of the latter.

Another criterion to distinguish the rendering algorithms lies in the scene representation. Distributed ray tracing and rasterization belong to the group of rendering algorithms which operate on geometrically represented scenes. On the other hand post-processing methods (together with point-based rendering methods), such as filtering and image-based ray tracing, operate on sample-based representation of the scene [12], eg. layered depth images [26].

Most methods make assumptions on the sensor orientation so that it might be hard to extend them to support tilt-shift configurations. The most flexible in those situations is the plain ray tracing.

For more detailed information on the various DoF rendering methods and camera models used in computer graphics the reader should also consult the existing surveys [2, 3, 10, 4, 19].

3.1 Monte Carlo ray tracing

Ray tracing methods estimate the radiant energy going to a pixel by sampling the radiance along incoming rays. In general the light transfer is recursively evaluated at the points of ray-scene intersection. Although the variants like distribution ray tracing [7], path tracing [16] and others differ in the strategy of tracing rays while evaluating the incoming radiance the ray generation is usually similar. Instead of a singular pinhole a more complex lens model is added between the sensor and the scene [17, 28]. Since the light paths have to pass the lens elements in a known order the complex lens system can be put outside the ordinary acceleration structures. See fig. 1d for an example image from our CPU implementation.

The important thing is that the lens model has a non-zero area which has to be sampled as well. In theory it is the image of the aperture stop, the exit resp. entrance pupil (when looking from the back or from the front). Those two pupils are defined only for on-axis rays. For off-axis rays in ideal thin lenses the pupil remains the same and can be sampled directly. However, in complex lens systems not only the aperture stop can block the light passage which results in the view-dependent effective pupil – projection of the visibility through the lens on a given plane. Sampling the precomputed effective pupil or at least its bounding circle leads to decreasing the amount of rays blocked inside the lens [28] and thus also the image variance. A simpler but not as efficient technique is to sample the whole surface of the outer lens element.

The circular pupils can be sampled by mapping samples from a unit square onto a unit circle with a suitable square to circle mapping [27]. For more complicated aperture shapes this can be combined with rejection sampling.

This approach can give the ground-truth results with no artifacts other than noise and it is thus considered the reference one.

3.2 Multi-view accumulation

The multi-view accumulation method [13] is based on the observation that each view through a single point on the entrance pupil of a thin lens is equivalent to some pinhole projection with an off-axis frustum [6]. Those pinhole views can be then rasterized by the GPU as usual. By

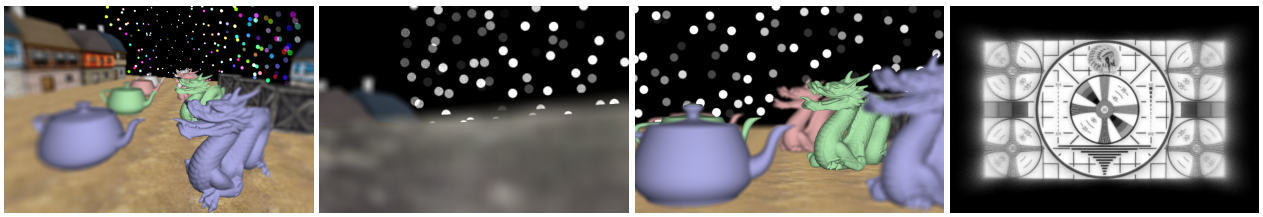


Figure 1: Example images from out implementation of various DoF rendering methods. Left to right: (a) image-based ray tracing with tilt-shift thin lens model, (b) image-based ray tracing handling partial occlusion with bokeh, (c) bokeh in multi-view accumulation, (d) sequential lens ray tracing with a biconvex lens.

sampling the entrance pupil and accumulating the rasterized views the image with depth of field is obtained. The key for the off-axis frusta construction is that they must intersect each other at the image of the sensor (on the focus plane).

The original method used the hardware accumulation buffer (with cca 12-bit integer precision). For accumulating thousands of views (to obtain correct bokeh) rendering to textures (eg. via OpenGL's Frame Buffer Objects) with at least 32-bit floating point precision and manual accumulation is needed [29]. This also allows incremental rendering where the intermediate results are displayed in real-time during the longer convergence. See fig. 1c.

The basic method is very simple but is limited to the thin lens model. Tilted configurations were shown to be supported in [5], unfortunately without any details, and are discussed in section 5. An extension to approximate complex lens systems is described in [14].

Advantages of this method compared to post-processing methods are that it displays all parts of the scene visible from the entrance pupil (not only from its center) and the visibility is already solved (by the z-buffer). A disadvantage is that the entrance pupil is sampled per-image, not per-pixel, so the convergence is slow.

3.3 Layers and their extraction

Before we can describe the DoF post-processing methods themselves we need to learn more about their input data.

In a pinhole image (perspective projection) only the parts of the scene which are directly visible from the center of projection can contribute to the output image. On the other hand for a lens with a finite aperture even parts of the scene which are occluded in the central pinhole view can become visible in other views and thus take part in the output image. Since both the directly visible and occluded parts of the scene cannot be represented in a single image, they must be stored in several layers.

Each image represents a 2D table of samples of the incident radiance function from the scene to the center of projection. Each sample might be then understood as a single light source. Except that the sampled color (or precisely radiance) from the scene is not enough for depth of field rendering since the effect of a light source on the image also depends on its depth. Thus each layer consists of a color image and a depth image. Usually the layers store the results of frustum transformation normalized to the $[0.0; 1.0]^3$ cube.

The sampled radiance is valid only for a single direction. Assuming that the exitant radiance of scene surfaces does not vary too much when changing the viewing direction a little the sampled radiance can approximate the true radiance from another viewpoint (on the front element of the lens) quite well. This problem can be solved with deferred shading [20] where surface properties are sampled and radiance from given viewpoint is computed later.

There are two approaches in extracting layers from the scene – depth interval layers [25, 1] and depth-peeled layers [11] – each with its pros and cons.

In depth-interval layer extraction the scene is divided into disjoint intervals of depth and each layer contains the surfaces visible in that layer. This results in that the whole images are ordered by depth which could be exploited in some methods.

On the other hand depth peeling [11] produces layers where each pixel is ordered by depth independently. The first layer contains what is visible directly, the second what is hidden after the first layer and so on. This results in fewer layers, since the number of layers is limited only by the depth complexity of the scene. The difficulty is that a patch of pixels from one surface might be interspersed in many layers.

In general the depth peeled layers provide a more compact representation than depth interval layers, since there are fewer empty areas. Thus a lesser number of layers is needed, saving some memory.

The layers can be rendered by a GPU scan-line rasterizer or with a ray tracer modified with additional depth checks, resp. taking k-th intersections instead of the first ones.

For accurate rendering of strong bokeh it is necessary that the color images in the layers are HDR images, eg. represented with floating-point numbers. The resulting output image might be then tone-mapped to LDR.

3.4 Image-based ray tracing

A recent technique to accelerate depth-of-field rendering via a combination of rasterization and ray tracing is the

image-based ray tracing [20, 21]. The main idea is to rasterize a part of the scene visible from the lens and use it as a sample-based representation of the scene for the ray tracing stage. Shading of the visible scene objects is done once and then reused for the views from the many lens samples. Thus the time complexity of the ray tracing stage depends on the image resolution not on the scene complexity. From this point of view the method is most suitable for very complex scenes.

Since the rasterization hardware only supports the ideal pinhole model the view-dependent scene representation is based on a single image from the center of the entrance pupil with the field of view approximated from the original lens model. From the other sample points on the entrance pupil might be visible some parts of the scene "around the corner", so that several layered depth images have to be extracted. The layers contain both color and depth images which are afterwards treated as height-fields and can be intersected instead of the original scene objects.

The two methods differ in the kind of layered depth images used, [20] works with depth-interval layers, while [21] is based on depth-peeled layers. Properties of the layers further determine the algorithms such as height-field intersection or layer extraction and possible camera models. We will describe the second approach. Example images from our GPU implementation of this method can be seen in fig. 1a and fig. 1b.

First the perspective frustum from the entrance pupil center needs to be found, which will be used for sampling the scene into the rasterized layers. For non-tilt configurations and the thin lens model the sensor image defines the frustum shape. Similarly for a shifted sensor an off-axis frustum is constructed. For a tilted sensor a technique referenced in chapter 5 might be used. More complex lenses have to be approximated by thin or thick lens models for this purpose.

Having the frustum matrix the layers can be extracted by depth peeling [11]. Compared to the original method the depth peeling process is easier on current hardware since it is possible to utilize floating-point render textures and programmable shaders. In short, in each iteration the previous layer's depth image is utilized as a secondary z-buffer with an additional depth test to discard the nearer geometry. In practice the depth peeling code can be prefixed to ordinary material shaders. OpenGL's Array Textures can be used for storing the layers. The depth images can be stored separately, packed by four channels into one image for more efficient texture lookup later.

The ray tracing stage then consists of ray generation, intersection with the height-field layers and color accumulation. In case the GPU does not support random number generation we need to provide them in a texture. In practice we tried a 3D texture of size $64^2 \times N$ samples (where N is the number of samples of a single pixel) without excessive artifacts from tiling. The chosen area on the lens should be sampled and the ray transfer within the lens model evaluated as in the ordinary Monte Carlo ray tracing. The rays need to be transformed by the frustum matrix to match the space of the height-field layers.

In the depth-peeled layers we cannot assume that the height-fields are continuous or in disjoint depth intervals. A robust technique for intersecting such height-fields is per-pixel traversal of the ray footprint, ie. the pixels under its orthographic 2D projection, along with a robust intersection test. The 2D version of the DDA algorithm for voxel traversal can be used with some modifications for more robustness in singular cases [29]. Since we treat the height-fields with nearest-neighbor interpolation the intersection tests must be extended by some epsilon tolerance. To reduce memory bandwidth the depth layers can be packed and the intersection test must be modified to work with 4-component vectors.

Several acceleration techniques have been proposed in the original method [21]. First, some geometry does not need be extracted into the layers since it can be shown that it is not visible from any point on the entrance pupil. This extended umbra depth peeling technique can lead to a high speedup, but on the other hand it assumes the thin lens model and makes the height-field treatment more complicated due to the introduction of undefined values. Second, the ray footprint traversal can be accelerated by iteratively clipping the ray extents with the knowledge of minimal and maximal height-field values under the ray footprint. Those values can be efficiently evaluated by N-buffers [9] constructed from the depth layers. Also the N-buffer queries can be done for multiple rays at once.

3.5 Filtering

The filtering approach is based on the concept of point spreading function (PSF) known from Fourier optics where the lens system is considered a linear system. The PSF is an impulse response of the system to a point light. Then the image on the sensor is given by the convolution of the exitant radiance function of the scene with the PSF. However, the model assumes no occlusion, so the visibility has to be solved by other means. Also the PSF is non-constant and depends many factors. When neglecting the wave-optics effects like diffraction the PSF is mostly affected by the aperture stop shape and directly influences the appearance of bokeh.

Most of the methods for interactive depth-of-field rendering are based on the convolution of some PSF kernel with a sample-based scene representation – a rasterized pinhole view. Many of them lead to physically incorrect results and artifacts. One reason is the lack of solving visibility, the other is in using some PSF kernels with an inappropriate convolution method.

It has been recently shown that for convolution with spatially varying kernels there is a difference and duality between gathering and spreading filters [19]. The image formation corresponds to spreading filters and using those PSF in the gathering context causes major artifacts (and vice versa).

For some restricted classes of PSFs there exist acceleration techniques (fast spreading filters) for reducing the convolution complexity from $O(n^2)$ to $O(n)$ or even $O(1)$ with respect to the rasterized kernel radius [19].

The visibility in context of filtering is usually solved by representing the visible part of the scene with multiple layers and alpha compositing them after being filtered. Since the layers are rendered by another technique such as rasterization or ray tracing the filtering methods are denoted as post-processing methods.

3.6 Comparison of ray tracing and filtering

We can compare the two approaches to DoF rendering. Ray tracing methods solve visibility correctly but have problems with optimal sampling. Filtering methods offer optimal sampling but have difficulties with solving the visibility within scene (occlusion).

Getting a large CoC sampled properly in ray tracing requires many samples to reduce noise to tolerable levels, while for in-focus areas a single sample might be sufficient. Unfortunately, for a given pixel on the sensor we are not aware of a method of efficiently generating lens rays ordered by decreasing contribution of light intensity. Eg. importance sampling successfully employed in image-based lighting [24] cannot be readily used due to possibly complex ray transformations within the lens and limited scene visibility from a pixel.

On the other hand spreading filters can rasterize the PSF according to the sensor resolution, spreading the light exactly to the affected pixels without noise present in Monte Carlo methods. The cost is that occlusion has to be solved by other means.

Another view is on supported PSFs. Ray tracing methods can support arbitrary lens models with various PSFs, but have to evaluate the propagation of rays in the lens systems which might be costly. Filtering methods are limited by the complexity of PSFs and their efficient representation, evaluation and spreading.

Ray tracing of the original scene and multi-view accumulation has also the advantage over image-based methods that they provide implicit anti-aliasing via multi-sampling. Methods working with discretized images eg. have no information of exact position of a highlight smaller than a pixel. Thus a bokeh pattern from such a highlight might be slightly translated in the image-based methods compared to more accurate results of the multi-sampling methods. Also CoCs from light sources outside the pinhole field of view might be missing there.

4 Complex lens representation

An ideal thin lens can be fully described just by its focal length and its aperture stop (a circle, polygon, raster bitmap, etc.). For real-world complex lens systems typically a simplified representation must be given. The traditional representation is followed in [17, 28] – a sequence of spherical caps or planar stops and subsequent materials. The ray transfer is evaluated by sequential ray tracing with analytical intersections and refractions. The complexity is $O(n)$ with respect to the number of elements. Given an incoming ray it is either transformed into an outgoing ray or absorbed inside the lens.

We present a conceptually novel representation of ray transfer behavior of lenses. Rays can pass through a lens either from front to back (along the optical axis) or in the opposite direction, this corresponds to forward and backward ray tracing. The ray transfer in either of the two directions can be treated as a mathematical function, a *lens ray transfer function* (LRTF), which maps incoming rays to outgoing rays and which is defined only for rays that pass. A single lens system can be described by various LRTFs depending on the particular parametrization of rays.

An LRTF can be defined either implicitly and evaluated procedurally as in sequential ray tracing or it can also be sampled into a table and evaluated approximately by interpolation. The ray transfer can be then evaluated in $O(1)$ time with respect to the number of lens elements. Precomputation of the LRTF can be utilized for optimizing the ray generation in both real-time and off-line rendering. Another benefit is the lack of need for specialized intersection routines for different types of lens elements in the evaluation stage. An interesting possibility is in measuring the ray transfer function from real world lenses without being aware of the internal design!

Given some assumptions we propose one of the LRTF parametrizations which we find useful. A ray in the 3D Euclidean space can be parametrized by a 3D position and a 3D direction. For the direction there are in fact only two degrees of freedom (eg. when using spheric coordinates). Similarly the ray position can be related to the outer surface of the lens which is assumed to be a smooth finite 2D surface (rays that do not intersect it cannot pass through the lens). The rays (both incoming and outgoing) can be thus parametrized by four parameters in total.

From the many possible ways of parametrizing the ray position we have chosen the hemispherical coordinates of the ray intersection with the bounding hemisphere over the aperture of the outer lens element surface (eg. the base circle of a hemispherical cap). It is independent on the exact shape of the outer lens surface and can be constructed even when only the aperture radius of the outer lens surface is known. Rotation around the optical axis is simple. The positions of the front and back lens apertures on the optical axis should be taken into account in the transformations for the bounding hemispheres.

In this parametrization we assume the range of ray directions is limited to a hemisphere pointing outwards from the lens. This should be no problem for the majority of lenses except for some fish-eye lenses. The obvious way to represent ray directions would be spherical coordinates, but their drawback is a singularity at the pole which results

in poor sampling of the most important region.

It is better to transform the point on the unit hemisphere (direction) to a unit circle via stereographic projection [8]. The rotation around the optical axis is still easy and the projection is without a singularity.

We define the ray direction to be represented relatively to the intersection position (the position on the unit circle is merely rotated). We can then denote the parametrized ray as (θ, ϕ, s_x, s_y), where θ and ϕ are declination and azimuth of the position and (s_x, s_y) are stereographic coordinates of the direction; $\theta \in [0; \frac{\pi}{2}]; \phi \in [0; 2\pi]; s_x, s_y \in [-1, 1]$.

It can be shown that for lenses rotationally symmetric around the optical axis the LRTF in this parametrization is also rotationally symmetric. In particular if we denote the LRTF as L it holds $L(\theta, \phi, s_x, s_y) = L(\theta, 0, s_x, s_y) + (0, \phi, 0, 0)$. This can be exploited to reduce the sampled table dimension to three, with the values remaining a 4-vector.

For the purposes of evaluating the function values from 3D texture we can rescale the parameter ranges to $[0; 1]$. In case of ray absorption the LRTF is undefined and this can be represented with a special value; to minimize discontinuities $(1, 0, 0, 0)$ might be a good candidate.

In summary, the ray transfer through a complex lens system might be described by a sampled LRTF table, radii of the front and back lens element apertures and their distance along the optical axis. The LRTF table can be precomputed and then utilized to speed up the ray generation phase of ray tracing. A ray has to be procedurally transformed to the parametric form before the LRTF evaluation and back to the standard form afterwards. The concept maps well to the current GPU architectures as each sampled function value can be precomputed in parallel and the evaluation is based on texture lookup and interpolation, the most optimized operations on a GPU.

5 Open problems

Depth of field How to define blur amount and depth of field for tilt-shift configurations? For thin lenses the CoC might be of an arbitrary conic section shape (including infinite-size hyperbolas).

Multi-view accumulation Although it was shown that this method can be extended to support tilt-shift configurations [5] no details were provided. We found the key is still that the frusta from the lens sample point intersect at the image of the sensor which is a general quadrilateral. In order to obtain a rectangular frustum for rasterization a proper orientation of the near plane must be found. A promising way of finding the correct transformation matrices is described in a method from the computer vision area [23].

The method approximating the image from a complex lens system with many blended pinhole views [14] should be tested again on the modern hardware and compared with the other methods. Anyway, it would probably have similar limitations as the original multi-view accumulation.

Image-based ray tracing A single pinhole frustum might not contain all parts of the scene visible from other viewpoints on the lens. This might be a problem especially for very wide angle lenses and also for near objects. The conditions when the error is significant should be examined.

The depth-interval layer decomposition is dependent on the CoC size which is well-defined only for non-tilt configurations. Given a blur metric for tilt-shift configurations could the method with this kind of layers produce correct results?

Obtaining layered depth images via ray tracing instead of rasterization in the depth-peeling phase should be explored. The sampled scene representation would still allow cheaper intersections for large scenes and rendering the layered depth images would be simpler for tilted configurations as there would be no need for computing suitable frustum matrices.

Lens ray transfer function Better parametrizations of the LRTF should be given. The one proposed here is limited by the range of ray directions and might waste some memory with larger undefined areas. As for the range of directions on a spherical cap larger than a hemisphere the stereographic projection is general enough to produce just a larger circle which can be rescaled to the unit size.

Also the practical LRTF application to accelerate ray generation in a ray tracer should be measured for accuracy and performance.

The LRTF might be measured from real-world lenses for which a method and an apparatus need be constructed. This could allow eg. to match the lens characteristics of a real film footage with synthesized images without knowing the exact internal lens design.

By changing the aperture stop size or shape only the domain of the LRTF is affected not the values. Thus a variable-sized diaphragm even with an asymmetric shape could be represented outside the LRTF table. Also the properties of the LRTF domain might be exploited for a more compact representation (eg. a boundary of a compact region).

Does the LRTF provide enough information for lens flare rendering or what additional information is needed to produce correct results without having the original lens design?

Can the LRTF model be extended to support movable element groups (zoom lenses, focusing by group movements) without an excessive memory usage?

Currently, this LRTF parametrization assumes ray transfer at a single wavelength which would lead to a lack of chromatic aberration. Would it be possible to exploit some

coherence to support wavelength dependence without increasing the dimensionality of the precomputed LRTF table?

Rendering with wavefront aberrations was shown in the filtering approach [1]. Could they be used to represent ray transfer in complex lenses? As the list of Zernike polynomial coefficients directly represents the amount of the various optical aberrations this representation might enable modifying the behavior of existing lenses and synthesizing new ones without the need to have a geometrical design. Also the memory consumption could be quite low.

Lens sampling Sampling of the effective pupil of complex lenses should be revised for real-time rendering. In [28] the pupil is precomputed for each pixel on the sensor which is valid only for a single sensor plane. A more compact representation of an approximate effective pupil function for an arbitrary sensor plane should be explored, along with its precomputation and usage for sampling. The assumption of rotation symmetry can again be used for dimension reduction.

Filtering and PSFs The usage of PSFs of complex lens systems with filtering methods (representation, precomputation, evaluation, etc.) should be further explored.

The comparison of the fundamental approaches leads to a question for a hybrid method of ray tracing and filtering which would offer optimal sampling while computing visibility easily. One of the possibilities might be in modifying the PSF kernel on the fly – invisible parts (decided by ray tracing) would be "cut off" from the rasterized PSF kernel, then it would be differentiated and spread.

Is it possible to modify spreading filters to work with physically correct PSFs of tilt-shift configurations? Note that the rotated sensor plane may result in infinite conic section PSFs even for a simple thin lens and also the radiometric situation with natural vignetting is more complicated.

Can PSFs in polynomial spreading filters be represented by orthogonal polynomials (eg. Zernike polynomials) with strictly limited values in order to suppress numerical precision problems?

It should be verified whether fast spreading filters are really compatible with solving visibility by per-pixel layers using depth-peeled image layers. Their advantage is in requiring fewer layers compared to depth-interval layers. Per-pixel layers [22] seem to be suitable for spreading filters since in the original method a kind of spreading was done, albeit by different means. Unfortunately, [18] only mention per-pixel layers without providing details whether they in fact used per-image or per-pixel layers.

Per-pixel layers do not seem to be fully compatible with precomputed rasterized PSF differences since some per-pixel computations has to be made. Note that for deciding the output for a pixel of the rasterized CoC we need to know the source and destination pixel depths. The information on source pixels cannot be restored during the phase of spreading of PSF differences. A way to combine those two methods which seems to be possible is the following procedure:

1. take the original rasterized PSF
2. for each of its pixels make decision to which output layer it should go, which produces three clipped PSFs
3. differentiate each of them
4. spread each of them into the corresponding layer

Ie. spreading could be done after deciding the output layer and for each part of the PSF separately.

6 Conclusion

In this paper we have presented the various approaches to interactive physically-based depth-of-field rendering and compared them in several aspects. We have proposed an alternative representation of the ray transfer behavior of complex lens systems which might be useful for accelerating the ray generation in ray tracing methods. Most importantly we have shown many open problems in the area of rendering with complex camera models which could serve as a basis for further research.

7 Acknowledgments

I would like to thank to Alexander Wilkie and Josef Pelikán for their support.

References

[1] Brian A. Barsky. Vision-realistic rendering: simulation of the scanned foveal image from wavefront data of human subjects. In *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, APGV '04, pages 73–81, New York, NY, USA, 2004. ACM.

[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 i, object-based techniques. In *ICCSA'03: Proceedings of the 2003 international conference on Computational science and its applications*, pages 246–255, Berlin, Heidelberg, 2003. Springer-Verlag.

[3] 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. In *ICCSA'03: Proceedings of the 2003 international conference on Computational science and its applications*, pages 256–265, Berlin, Heidelberg, 2003. Springer-Verlag.

[4] Brian A. Barsky and Todd Kosloff. Algorithms for rendering depth of field effects in computer graphics. *Proceedings of the 12th WSEAS international conference on Computers 2008*, 2008.

[5] Brian A. Barsky and Egon Pasztor. Rendering skewed plane of sharp focus and associated depth of field. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 92, New York, NY, USA, 2004. ACM.

[6] Paul Bourke. Offaxis frustums - opengl. July 2007.

[7] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18:137–145, January 1984.

[8] H.S.M. Coxeter and H.S.M. Coxeter. *Introduction to geometry*. Wiley Classics Library. Wiley, 1989.

[9] Xavier Décoret. N-buffers for efficient depth map query. *Computer Graphics Forum*, 24(3), 2005.

[10] Joe Demers. *GPU Gems*, chapter Depth of Field: A Survey of Techniques. Addison-Wesley, 2004.

[11] Cass Everitt. Interactive order-independent transparency, 2001.

[12] Markus Gross and Hanspeter Pfister. *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[13] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 309–318, New York, NY, USA, 1990. ACM.

[14] Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. An image-based model for realistic lens systems in interactive computer graphics. In *Proceedings of the conference on Graphics interface '97*, pages 68–75, Toronto, Ont., Canada, Canada, 1997. Canadian Information Processing Society.

[15] Matthias Hullin, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee. Physically-based real-time lens flare rendering. *SIGGRAPH 2011*, 2011.

[16] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20:143–150, August 1986.

[17] Craig Kolb, Don Mitchell, and Pat Hanrahan. A realistic camera model for computer graphics. 2002.

[18] Todd Kosloff, Michael Tao, and Brian A. Barsky. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. *Graphics Interface 2009*, 2009.

[19] Todd Jerome Kosloff. *Fast Image Filters for Depth of Field Post-Processing*. PhD thesis, EECS Department, University of California, Berkeley, May 2010.

[20] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (Proc. ACM SIGGRAPH ASIA)*, 28(5):1–6, 2009.

[21] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. *ACM Transactions on Graphics (Proc. ACM SIGGRAPH'10)*, 29(4):65:1–7, 2010.

[22] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using point splatting on per-pixel layers. *Computer Graphics Forum (Proc. Pacific Graphics'08)*, 27(7):1955–1962, 2008.

[23] Kok-Lim Low and Adrian Ilie. View frustum optimization to maximize object's image area, 2001.

[24] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[25] Cary Scofield. 2 1/2—d depth-of-field simulation for computer animation. pages 36–38, 1992.

[26] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 231–242, New York, NY, USA, 1998. ACM.

[27] Peter Shirley and Kenneth Chiu. A low distortion map between disk and square. *journal of graphics, gpu, and game tools*, 2(3):45–52, 1997.

[28] Benjamin Steinert. Simulation of real photographic phenomena in computer graphics. Master's thesis, Universitat Ulm, 2009.

[29] Bohumir Zamecnik. Interactive preview renderer for complex camera models. Master's thesis, Charles University in Prague, Faculty of Mathematics and Physics, December 2011.

Simulation of Camera Features

Michal Kučiš*
Supervised by: Pavel Zemčík†

Faculty of Information Technology
Brno University of Technology
Brno / Czech republic

Abstract

Computer vision algorithms typically process real world image data acquired by cameras or video cameras. Such image data suffer from imperfections caused by the acquisition process. This paper focuses on simulation of the acquisition process in order to enable rendering of images based on a 3D generated model that are as close to the images acquired by cameras or video cameras as possible. Its main purpose is simulation of video input for computer vision applications, e.g. robot navigation. Imperfections, such as geometry distortion, chromatic aberration, depth of field effect, motion blur, exposure automation, vignetting, inner lens reflections, and also imperfections caused by image sensor features are considered.

The paper, besides description of imperfections of the acquisition process and description of imperfections simulation, also presents results of the simulation software through illustrative figures produced by the software.

Keywords: Camera Imperfections Simulation, Depth of Field Effect, Distortion, Motion Blur, Vignetting, Lens Flare, Image Sensor Features

1 Introduction

Computer vision is a field of computer graphics that allows for acquiring, analyzing, and understanding images. An input of computer vision algorithm is a set of images, which are typically captured by camera or video camera. An output is a symbolic information, e.g. information about identity of subject in the image or any other valuable information. Input images are not perfect copies of the reality since they are affected by many camera features. A "perfect" image of the real world is modified and camera features add imperfections to the image. The imperfections strongly influence success of the algorithms, so in general, the algorithms need to take into account these imperfections.

Implementations of computer vision algorithms in real world application work with images or sequences of images, which are usually captured by a camera. Such can be e.g. a security camera or a robot camera. Also, the implementations have to be trained and tested on a set of images. The more similar the test images and the images captured in the real world application are, the more precise the results of algorithms are.

The best way to acquire similar data is to use a camera, which would be used in the real world application. However, this option is often either expensive or unavailable. The second option is to use a dataset of images, which was acquired by a different camera or cameras. The images would be affected by different camera imperfections, so the result would be worse. The third option is to generate data by computer and use it as input images. The generated images and dataset of images are generally cheaper, but the results of such solution would be worse as well.

A good way how to get better results is to modify computer generated "perfect" images to look similar to the images created by a target device. This paper describes a simulation method for acquiring target device like images.

2 Related work

No complex simulator with all the desired features was known up to the date, but there are relatively many publications that describe the camera features and simulation methods for some of the features [4, 5, 8, 10].

Distortion is a general problem of lenses. Measurement and correction of the distortion are described in [12]. Simulation of distortion is just a reversed process. Detection and elimination of a chromatic abberation (a form of distortion) is shown in [5].

Depth of field effect problem is widely explored. Many algorithms that simulate the effect are known. Basic description of these methods can be found in [3, 6, 10]. Detailed description of the interactive depth of field diffusion method is described in [2]. This method is interesting because it does not suffer from some problems that appear in the other post-processing methods.

Many camera sensors capture color images using a color filter array. Description of this feature and its effect to the output is described in [11]. Another problem of the sensors is noise. Noise evaluation of CCD sensors is shown in [9].

*xkucis00@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

Characteristics of many cameras and sensors are described by EMVA Standard 1288 [1].

3 Description of simulation

The proposed algorithm consists of application of several effects, which simulate the described features. All the simulated effects are described in the following sections and every one can be adjusted by using parameters. Default order of the effects application is shown in Figure 2. (In the implemented software, the parameters of the effects and their order are described with configuration file.) Input of the simulator is a (high-dynamic range) color image, a depth map, and an environment map. The color image is a "perfect" image that will be modified by the simulator. The image is shown in Figure 1. The depth map defines distance between a pixel in real world and a camera. This map is used by the depth of field effect and the motion blur effect.

Figure 1: Input color image; this image is modified by the simulator

4 Lens features

This section describes lens features: distortion, chromatic abberation, vignetting, and lens flares. The camera lens is an optical system that affects more phenomena like spherical aberration, astigmatism, coma and more, but in general their influence in the camera lens is not considered or it is rarely considered. Therefore, we have not included these phenomena.

4.1 Distortion

The camera image is a 2D-projection of the real world. In optics, distortion is a deviation from rectlinear projection. This projection guarantees straight lines remain straight after the projection.

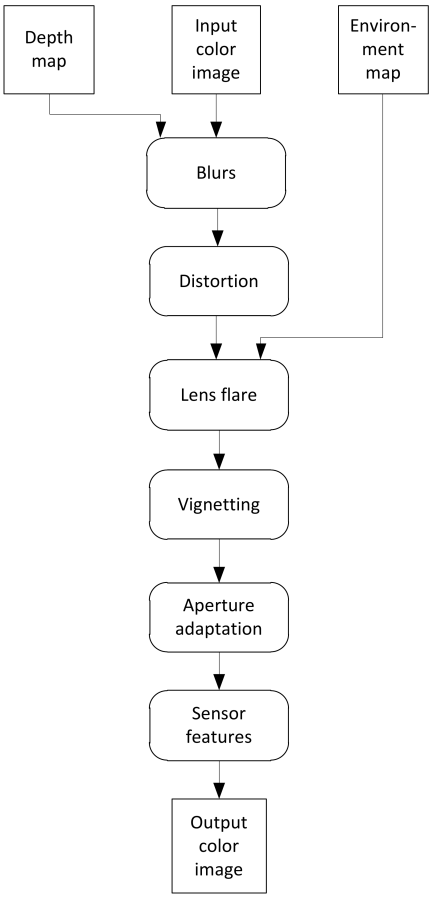


Figure 2: Algorithm of the simulation. The effect 'Blurs' links up depth of field effect, motion blur, and structural blur. The effect 'Sensor features' links up noise and color filter array.

The most commonly encountered distortion is radially symmetric distortion. This type of distortion arises from the symmetry of the camera lens and the symmetry of the camera optical system. In this case, the distortion can be simulated by the radial distortion model. The radial distortion model uses 6-7 real-valued variables which specify the distortion. [12] contains a description of this model.

Radial distortion model can be expressed as:

$$r_d = f(r), \tag{1}$$

where: r_d – destination distance of input pixel position
r – source distance
f – distortion function

If the distortion is not symmetric, it cannot be simulated by the radial distortion model. In that case, warping is used that as described in [7, 14].

The distortion simulation uses a 2D filter to remap the input pixels to the output pixels. For better results, we use the Lanczos filter in the current version.

4.2 Chromatic abberation

Chromatic abberation is a type of distortion in which the lens is not able to focus all colors to the same convergence point. This distortion is caused by different refractive indexes for different light wavelengths. The distortion is mainly observed in the edges between a bright light and a shadow.

The abberation can be simulated by modification of the distortion model. Every color channel of the input image is deformed by a slightly different distortion. The result is a color contour in the edge of the bright and dark areas. The abberation is radially symmetric. Therefore, it can be implemented by radial distortion model and also by warping.

We simulate the abberation by:

$$r'_{color} = f_{color}(r), \tag{2}$$

where: r'_{color} – distance between the output pixel position and a center of the image
r – distance between the input pixel and the center
$f_{color}(r)$ – central distortion model for $color$

4.3 Vignetting

Vignetting is a reduction of an image's brightness at the image periphery. The vignetting is mainly radially symmetrical, but rarely it can be also radially asymmetrical. The simulation of this feature is straight-forward. Pixels of the input image are multiplied by the pixels of a vignetting mask image. The vignetting mask is defined by the used camera and can be obtained by measuring. It can be measured in a scene where only one solid color occurs (e.g. a white paper). In this case, an image brightness is the vignetting mask. The example of the vignetting is shown in Figure 3.

Vignetting is calculated as:

$$B(x,y) = m(x,y)S(x,y), \tag{3}$$

where: B – output image
m – vignetting mask
S – input image

If it is centrally symetric vignetting, m is computed:

$$m(x,y) = m'\left(4\frac{(x-sx/2)^2+(y-sy/2)^2}{sx^2+sy^2}\right), \tag{4}$$

where: (sx,sy) – image size
m – vignetting mask

4.4 Lens flare

Lens flare is an unwanted light in lens system caused by inhomogeneities in the lens. The source of the lens flare are

Figure 3: Vignetting; the corners are darker than the center

unwanted internal reflections or scattered reflection inside the lens. It is difficult to describe all phenomena and their effects to the output image due to complex construction of the lens. Every lens creates different artifacts. Even the same type of the lens can create different artifacts under different conditions. In addition, the different position of the light source in the image can create different artifacts as well.

Lens flare manifests itself as a haze across the image or as visible artifacts. The haze can be simply simulated by adding color to all pixels in the image. The visible artifacts can be caused by a reflection on the aperture, inner reflections in the camera lens, refraction on inhomogeneities in the lens etc. Bright light source can create a star or can be mirrored etc.

A simple simulation of the lens flare artifacts is shown in Figure 4. We simulate this effect in following way:

$$B = S * K, \tag{5}$$

where: B – output image
S – input image
$*$ – convolution
K – convolution kernel

In Figure 4, a mirrored ghost is shown in the red circle. The ghost is created by:

$$B(x,y) = S(x,y) + \alpha_{xy}S(sx-x, sy-y), \tag{6}$$

where: B – output image
S – input image
α_{xy} – intensity of the ghost
(sx,sy) – size of the image

5 Aperture features

This section describes the simulated aperture features that affects the output image. It describes mechanisms of the

Figure 4: Lens flare effect; a bright light causes lens flare, the mirrored ghost of the light is shown in the circle

aperture adaptation and depth of field effects. This section also describes structural blur. This imperfection is not an aperture feature, but it can be simply simulated by a modified depth of field effect.

5.1 Aperture adaptation

Image brightness is influenced by three parameters in the real world application: an exposure time, a film speed and a f-number. The exposure time affects the motion blur. The film speed affects the sensor noise. The f-number describes a radius of the aperture. Moreover, the f-number affects the depth of field effect. With growing f-number, the depth of field effect becomes more visible (we suppose the f-number in form $f/1.2$, $f/2$, etc.). To calculate the output pixel color, we use these parameters in the following equation:

$$y_{ij} = x_{ij}\, c\, \frac{t\, s}{b^2}, \tag{7}$$

where: y_{ij} – output pixel color
x_{ij} – input pixel color
c – constant
t – exposure time
s – ISO film speed
b – f-number

The aperture adaptation calculates the f-number to get equal average grayscale value of image and middle gray. Middle gray is the universal measurement standard in photographic cameras and it stands a tone that is about half way between black and white. We compute an average image brightness by:

$$y_{avg} = \frac{\sum_i \sum_j w_{ij}\, x_{ij}}{\sum_i \sum_j w_{ij}}, \tag{8}$$

where: y_{avg} – average color
x – input image
w_{ij} – weight of the pixel

In general, pixels of our interest, e.g. in the center of the image, have bigger weight than the other pixels.

We implement two ways of adaptations. First one is in form:

$$b_{new} = \sqrt{y_{avg}}, \tag{9}$$

where: b_{new} – f-number
y_{avg} – average color of the image

The second method works iteratively. The drawback is, that the whole simulation has to be performed multiple times to get a valid f-number. On the other hand, the method usually converges to the true f-number. In addition, this method allows separate computation of average color and adaptation. The average color can be computed from the simulation input or the simulation output. This method is performed via a PID controller. The controller calculates an error value as the difference between the average color and the middle gray and it adapts the f-number to minimize the error. The formula is:

$$b_n = b_{n-1} + Kp\sqrt{e_n} + Ki \sum_{j=0}^{n-1} \sqrt{e_j}, \tag{10}$$

where: b_n – n-th f-number in the iterative computation
Kp – proportional gain
Ki – integral gain
e_n – difference between the middle gray and the average color with f-number equal b_n

5.2 Depth of field effect

Computer graphics methods for 3D scenes rendering typically use a pinhole camera model. The model leads to rendering entire scene in perfect focus. An image in the real world application is formed in an optical system where the light from a point in the scene converges at only one depth behind the lens. This depth is not necessarily equal to the sensor depth (Figure 5). The point in the real world appears spread over a region in the image. The region is called the circle of confusion (CoC). A computation of the circle of confusion is described in [3]. Simulation techniques of depth of field effect can be divided into object space methods and image space methods. The object space methods operate on a 3D scene representation. In general, the object space methods create more realistic results than image-space methods. The image-space methods operate on a 2D image of the scene. The image-space methods are postprocessing filters. The image is blurred with the aid of a depth map.

The simulator processes only 2D images, thus we are just interested in the image-space methods. We use two methods. The first method is based on the 2D linear filter

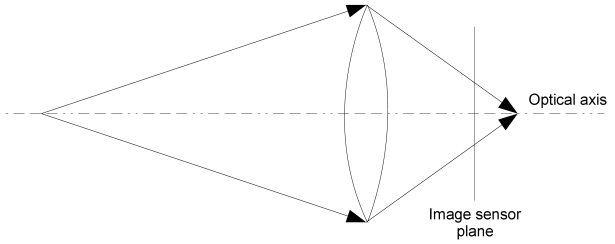


Figure 5: A point in the scene is projected as a disc on the image sensor plane, leading to depth of field effects

and it is called "Reverse-mapped Z-buffer depth of field". This method is shown in [6]. We use:

$$B(x,y) = \sum_i \sum_j psf(x,y,i,j)S(i,j), \tag{11}$$

where: B – output image
S – input image
psf – point spread function

The psf function is a point spread function, which relies on the circle of confusion computation.

The second implemented method is the depth of field using simulated diffusion (Figure 6). This method is based on principles of heat diffusion in an anisotropic environment. The algorithm is performed in two separate phases. In first phase, the diffusion is performed for each separate line. We create following equation for every pixel in the line:

$$y_i - x_i = \beta_{i+1}(y_{i+1} - y_i) + \beta_i(y_{y-1} - y_i), \tag{12}$$

$$\beta_i = min(CoC_{i-1}^2, CoC_i^2), \tag{13}$$

where: CoC_i – circle of confusion for i-th pixel
x_i – i-th input pixel
y_i – i-th output pixel

We get system of the equations that describes diffusion in the line. Then, we compute diffusion for every line of the image. In second phase, we compute diffusion for each column in the same way. This method is described in [2].

5.3 Structural blur

A perfect lens is able to project a point in the real world to an image point. A real world lens is not able to focus a real world point into an image point; therefore, the image of the point is blurred. We call this phenomenon the structural blur. We simulate this feature by modification of the depth of field effect computation. We just substitute circle of confusion computation by a structural blur ratio. Output of the structural blur is shown in Figure 7.

Figure 6: Depth of field effect; the camera is focused to the roof

Figure 7: Structural blur effect; the corners are more blurred than the center

6 Motion blur

In real world application, when a camera creates an image, the scene captured by the camera is not always static. Changes in the scene during the exposure are recorded to the image. Moving objects are blurred along their relative motion. This effect is called motion blur. Motion blur can be also caused by a camera motion. In such case, moving objects can be blurred and static objects must be blurred (the blur occurs along the change of the camera's viewport). An example of motion blur is shown in Figure 8. Motion blur is affected by the exposure time. Longer exposure time causes bigger blur effect.

The simulator expects to get input images representing of static scene with no motion blur even if the objects are in motion. The simulator uses fullscreen motion blur based on the algorithm in [13]. It allows us to simulate a camera motion.

Figure 8: Motion blur; the motion of the camera's view is simulated

We compute motion blur by a following equation per every pixel:

$$out = \frac{1}{n}\sum_{i=0}^{n-1} S\left(x_s + x_r\frac{i}{n-1}, y_s + y_r\frac{i}{n-1}\right), \quad (14)$$

$$x_r = x_d - x_s, \quad (15)$$

$$y_r = y_d - y_s, \quad (16)$$

where: out – output color pixel
n – number of steps
(x_s, y_s) – source coordinate of motion
(x_d, y_d) – destination coordinate of motion
S – input image

7 Sensor features

An output image is influenced by a sensor and its features. These features are noise, non-uniform response, and color filter array with demosaic filter. There are more features like blooming and artefacts caused by a transport in the sensor. In the current version, we simulate noise and color filter array with demosaic filter. Parameters of these features are known for many cameras unlike the other features.

7.1 Noise

Image noise is a random variation of brightness in images. The noise fundamentally limits the distinguishable content in the images. More information about CCD noise and evaluation of CCD sensor noise can be found in [9]. European machine vision association has also published EMVA Standard 1288[1] that describes the method of measurement and description noise of sold sensors and cameras. Simulated noise is shown in Figure 9.

In current version, we simulate noise the using:

$$B(x,y) = S(x,y) + r(f(S(x,y))), \quad (17)$$

Figure 9: Noise; temporal noise is added to the image

where: B – output image
S – input image
$r(i)$ – random number generator (e.g. Gaussian deistribution with a standard deviation i)
f – signal to noise ratio function

7.2 Color filter array

Most modern digital cameras acquire images using a single sensor overlaid with a color filter array. Each pixel on a camera sensor contains photo elements. The elemenents are monochromatic light sensitive and they do not distinguish wavelength of light. The output of the sensor is monochromatic image. Therefore, a color filter array is positioned on top of the sensor to filter out the component of light by the wavelength. The very common filter is the GRGB Bayer filter.

We simulate color filter array effect in two steps. We create a color mosaic image:

$$B_{color}(x,y) = S_{color}(x,y) \cdot M_{color}(x,y), \quad (18)$$

where: B – color mosaic image
S – input image
M – mosaic mask

The result of the color mosaicing is demosaiced by the following process:

$$B_{color} = S_{color} * K_{color}, \quad (19)$$

where: B – output image
S – input image
K – convolution kernel

If we want to simulate GRGB Bayer filter array, we use:

$$M_{red}(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \quad (20)$$

$$M_{green}(x,y) = \begin{cases} 1 & \text{if } x+y \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

$$M_{blue}(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ is even} \\ 0 & \text{otherwise} \end{cases} \quad (22)$$

$$K_{blue} = K_{red} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix} \quad (23)$$

$$K_{green} = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 1 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix} \quad (24)$$

In this case, simulated GRGB Bayer array is demosaiced by linear interpolation.

8 Conclusions and future work

This paper presents simulation of camera imperfections applied to computer generated images. The purpose of the simulation is to get computer generated images with features close to the features of images captured by real cameras. Such images can be used for image processing and computer vision applications testing. Some of the simulated imperfections can also be applied to high quality camera images to simulate output from lower quality cameras. The simulation of the camera imperfections is complex and presented solution still can be improved. Some of the features, such as distortion, chromatic abberation or vignetting can be simulated successfully. On the other hand, some others, such as lens flare, are very difficult to simulate because every camera has slightly different lenses that require individual and rather complex model.

Current version of the simulator is capable to process 0.9 frames per second. The test was performed with use of Intel(R) Core(TM)2 Duo P7350 2GHz on the image with resolution 640x480. In the future, we will optimize some of the algorithms, expecting an increase in performance.

Future versions of the simulator will allow simulation of more features, such as CCD sensor blooming, CCD streaking and more. Future works will also include modification of noise model. In order to improve the simulation, we will modify the implementation of the lens flare and we also will use more complex models of some imperfections.

9 Acknowledgements

I would like to thank Pavel Zemčík, Michal Košík, Peter Kubica, and Martin Krba for language corrections and valuable remarks. I also want to thank Oliver Zender from the Autrian Insitute of Technology for remarks and reviewing the article. This work was supported by the Artemis JU project R3-COP, grant no. 100233.

References

[1] *EMVA Standard 1288 - Standard for Characterization of Image Sensors and Cameras*. European Machine Vision Association, 3rd edition, 2010.

[2] John Owens Aaron Lefohn. Interactive depth of field using simulated diffusion. Technical report, 2006.

[3] 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.

[4] Brian A. Barsky and Todd J. Kosloff. Algorithms for rendering depth of field effects in computer graphics.

[5] Soon-Wook Chung, Byoung-Kwang Kim, and Woo-Jin Song. Detecting and eliminating chromatic aberration in digital images. In *Proceedings of the 16th IEEE international conference on Image processing*.

[6] Randima Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[7] Paul S. Heckbert. Fundamentals of Texture Mapping and Image Warping. Master's thesis, University of California, Berkeley, the United States of America, 1989.

[8] Evžen Hruška. *Fotografie na malý formát*. Státní nakladatelství technické literatury, 1959. L16-A-4-II/6227. (In Czech).

[9] Kenji Irie, Alan E. McKinnon, Keith Unsworth, and Ian M. Woodhead. A technique for evaluation of ccd video-camera noise. 2008.

[10] Todd J. Kosloff and Brian A. Barsky. Three techniques for rendering generalized depth of field effects. In *Proceedings of the Fourth SIAM Conference on Mathematics for Industry: Challenges and Frontiers (MI09)*.

[11] Xin Li, Bahadir Gunturk, and Lei Zhang. Image demosaicing: A systematic survey.

[12] Lili Ma, Yangquan Chen, and Kevin L. Moore. A new analytical radial distortion model for camera calibration. *CoRR*.

[13] Hubert Nguyen. *Gpu gems 3*. Addison-Wesley Professional, 1st edition, 2007.

[14] J. Žára, B. Beneš, J. Sochor, and P. Felkel. *Moderní počítačová grafika*. Computer Press, 2005. (in Czech).

HDR SMISS – Fast High Dynamic Range 3D Scanner

Tomáš Kovačovský*
Supervised by: Mgr. Ján Žižka†

Faculty of Mathematics, Physics and Informatics,
Comenius University
Bratislava / Slovakia

Abstract

Nowadays, increasing interest in computer graphics leads to higher demand for digital 3D models. It motivates the research in area of automatic 3D reconstruction. We present a system SMISS (Scalable Multifunctional Indoor Scanning System) based on structured light projection for automatic 3D reconstruction in metric space. We address the problem of low dynamic range of similar systems, which leads to incorrect measurements and we propose a novel approach to scan high dynamic range scenes with just a constant increase in scanning time using simple additional hardware. Our designed method uses the dynamic range of digital projector to suppress a scanning scene dynamic range. In result, our setup can be used as a flexible tool for future improvements of structured light scanning systems.

Keywords: 3D Scanning, HDR, Metrology, Reconstruction, Triangulation, Polarization, Multiple View Geometry, Computer Vision, Projector, Camera, Structured light

1 Introduction

The capability of contemporary computers, smart phones and tablets leads to an increase in popularity of the 3D content, which is now accessible in real time, thanks to increasing graphic performance. To satisfy this trend, we need a fast and easy enough method for creating the 3D content. Manual tools are slow and require skilled and trained designers, so there are a lot of methods for automatic 3D reconstruction of real objects. For extensive overview of methods and systems for automatic 3D reconstruction, we recommend [1].

For a long period of time, laser scanning has been an industrial standard for automatic 3D reconstruction and metrology. However, technology leap in modulating light trough digital projectors leads to method called structured light scanning. This method is becoming more popular even in industry, because of its flexibility and speed.

Despite all efforts, a system capable of full 3D reconstruction producing physically correct renders is still not available. To reach this stage, a lot of problems need to be solved. Significant limitation of current systems is an insufficient dynamic range for the general scene. This limitation is caused by bounded dynamic range of a digital camera, which is an essential part of structured light scanning systems. In addition, objects composed of highly contrast materials are common, because of pleasing visual appearance.

To address consulted needs and to deal with commented limitations, we offer:

- Flexible and easy to use 3D scanner SMISS, based on Gray Coded structured light, capable of fully automatic point cloud reconstruction.
- Innovative approach to increase DR of structured light scanning systems, using new hardware and algorithms.
- Optical co-axial setup, which can be used for future improvements consulted in chapter 8.

2 Related work

The first inspiration for our work was caused by a paper [2], in which authors created cost-effective system for scanning of non-colored objects. They used color coded Gray patterns projected by a digital projector and captured by a conventional camera.

Similar principle, the Gray coded patterns, was also used in [3], where authors offer tutorial for creating simple 3D scanning device.

Considerable effort has been invested in development of phase-shifting methods, we recommend [4], [5].

In [6], [7] authors separate specular and diffuse light components using polarization of the projected light, as well as the light captured by the camera.

Scanning under strong inter-reflection is a topic of a paper by [8], where authors use special logically coded pattern to negate the effect of short and long range inter-reflections.

Similar hardware setup, which we implement in this paper, was also used in [9] for capturing the geometry and reflectance and in [10], where authors used projector-camera setting as a tool for optical computing of matrix vector products in Krylov subspace to approximate the transport matrix for relighting.

* tomas.kovacovsky@gmail.com
† jzizka@gmail.com

3 Theory

3.1 Geometry

Every system based on structured light scanning uses a theory from projective geometry, triangulation, pinhole camera model, multiple view geometry and correspondence between camera and projector image space. This knowledge can be found in [11].

In our system, we work with analytical model of real camera, which also describes a lens distortion. For this purpose, the camera and projector have to be calibrated. We use method found in [12].

3.2 Lighting

During 3D scanning, the captured object is lighted by 3 types of light.

Surrounding lighting originates from light sources, which do not belong to the system.

Direct lighting is the part of light energy, which leaves projector and is directly reflected to the sensor after first reflection.

Indirect lighting is the remaining radiance measured by the camera. It could be caused by additional reflections, subsurface scattering and more.

Image captured by camera can be formalized as an addition of these 3 images, corresponding to different type of lighting. Surrounding light could be suppressed by subtracting frame with projector light turned off. To separate direct and indirect (global) components, we use [13] method.

3.3 Structured light

The structured light reconstruction is based on spatial geometrical coding of space trough geometrically coded patterns projected by projector. There are 2 main groups of patterns, phase shifting and binary. Both are fringe patterns. It means that the patterns are 2D images build from vertical or horizontal fringes. So in one direction, the patterns are constant. In second direction, the patterns are defined by 1 dimensional functions.

In the **phase shifting** patterns, the 1D function is often sinus with specified period. The goal is, to determine phase and period number for camera pixels, which correspond with part of an object surface lit by the pattern.

The **binary** patterns consist of fringes. Every fringe of the pattern has an index (vertical/horizontal position in image). Every index is coded into binary vector, so we need a vector size $n = \lceil \log_2 k \rceil$, where k is the total fringe count (resolution of the projector). Than we construct n patterns to code individual bits of binary vectors. In practice, the best general binary coding for this purpose is **Gray mirror coding**. For example, projector with image resolution 1024x768 has 1024 vertical fringes, so we need 10 bits to code the fringe index. This means, we create 10 patterns, 1 for every bit of information. We offer figure 1 for better understanding. In figure 3, there is scanned face under illumination of 3 different Gray coded patterns.

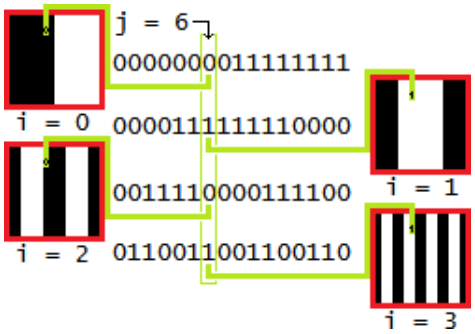


Figure 1: Gray mirror patterns for 16 fringes

HDR

In this paper, we present a novel method to reconstruct high dynamic range scenes. To understand the methodology of conventional HDR capturing, we recommend the book [14].

3.4 Our methods

In our solution, we develop a method for shadow detection and detection of highly reflective surfaces. We designed a fast iterative algorithm for compensating projector optical distortion. We also made methods for decoding Gray binary patterns using direct/global light separation, adaptive thresholding and negative image comparison. These methods could be found in [15].

4 SMISS

System SMISS is our implemented solution, which uses combination of digital camera and projector to capture object geometry. It shares the same geometrical principles as a two camera stereo setting, but instead of one camera, we use a digital projector. You can see schematic drawing and our most recent prototype in figure 2.

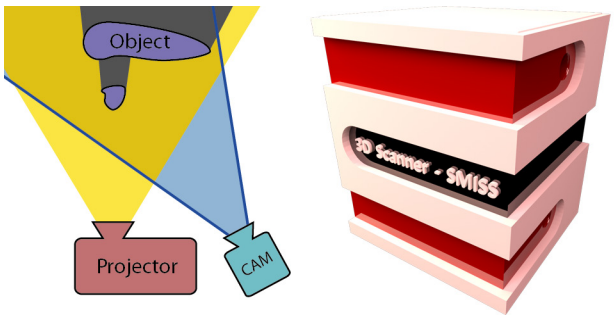


Figure 2: Schematic draw (left) and built prototype (right)

With the projector, we can control the light flow in the scene, projecting Gray binary patterns to spatially code the space. With the camera, instead of finding natural features of the object texture, we decode projected information. This means, we just need to receive enough information from projector, and we are not dependent on the object texture.

4.1 Calibration

The first important step in 3D reconstruction is the calibration process. In system SMISS, we use our designed easy to use calibration method. The whole calibration is done at once, using planar check board patterns with a known size. The full calibration requires a few images of calibration pattern in different position and rotation in space. In every iteration of calibration process, we found check board corners positions in the camera image. We than use Gray binary coded patterns to determine corners position in projector image space.

After the calibration process, we compute intrinsic camera matrices for the camera and the projector, which is inverse camera model and has the same optical principle. We are now able to compute matrix of extrinsic parameters to specify relative position and orientation between the devices in metric space. Calibration parameters are constant, as soon as the position of the camera and the projector is fixed and optics does not change.

4.2 Scanning

After the calibration, the system allows us to start a scanning program. The system SMISS implements the Gray Binary Code Scanning program, which allows pixel precision output.

In the scanning pipeline, in each iteration, specific structured light pattern is projected and the scene is captured by the camera under its illumination.

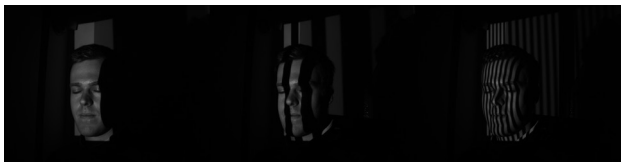


Figure 3: Scanned face

The full scanning pipeline consists of capturing black frame for surrounding light suppression. Then the mentioned method for direct/global light separation is used for shadow detection and detection of highly reflective surfaces. Camera pixels, which correspond to these areas, are omitted from the processing. After this, the Gray code pattern sequence is projected.

In the main loop, we decode the binary vector for every camera pixel (1 bit for a frame), which gives us correspondence between camera pixels and projector fringes. If we consider the image pixels as atomic parts, this uniquely defines curved plane (projector fringe projection) and line in space (camera pixel projection). In our situation, those have 1 unique intersection. The coordinates of this intersection defines object surface point.

Whole measurement, with texture information is then stored for a future use.

4.3 Limitations

To ensure the correct behavior of algorithms, we need to set a right camera exposure for the scanning. This means, that camera image cannot be overexposed, so we have to adjust exposure to fit highest response values. Because of bounded camera dynamic range, the low response areas of a scene could not be measured correctly (they are on a level of noise). For correct behavior, we need a sufficient signal to noise ratio for camera pixels. We offer HDR image of tripod wheel in figure 4.

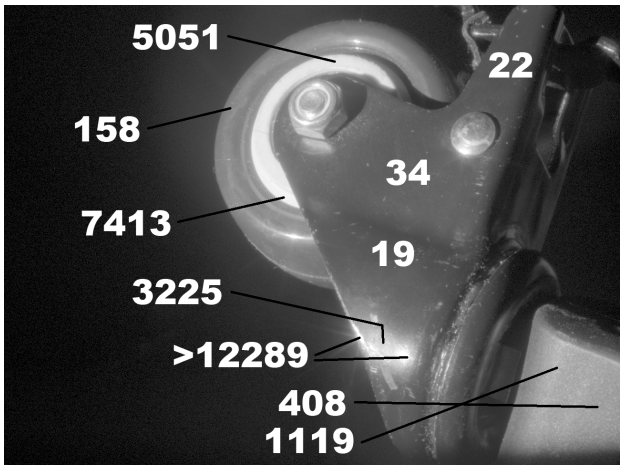


Figure 4: HDR camera image values captured with HDR SMISS. We use logarithmic mapping for visualization

Other limitation is caused by inter-reflections. We are able to detect highly reflective surfaces and dispose them from processing. These highly reflective surfaces, in our meaning, are mirror like surfaces or glossy surfaces with strong specular component. These attributes cause a small dependency on direct reflection and a strong dependency on global light reflection. This means that these surfaces could be more affected by light reflected from surrounding geometry, than by light directly from projector. Inverse problem of these materials is caused by focus reflection of projector light energy to other surfaces. Even if we detect this surfaces in camera image, the projector still send light to this surfaces, so those affect surrounding geometry a lot.

5 HDR SMISS

To address the problem of low dynamic range, we could use standard method of more expositions of the camera [16]. This method is popular for capturing HDR images in photography or for capturing environment maps for

relighting. This method has several disadvantages and limitations:

- It needs to be applied for every projected pattern, so we would need ***n***-times more frames to gather (***n*** is in respect of scene DR).
- Blooming effect caused by light scattering due to imperfect optics and CCD Blooming in highly contrast parts of the image.
- Does not offer solution for problems with reflective surfaces.

To get rid of those disadvantages and limitations of the standard method, our innovative solution uses combination of camera's and projector's DR. During the scanning process, in every projectors frame, every pixel of the camera image gathers the photons from the same area. This area is lighted by a set of projector's pixels.

For a camera pixel, let the integral of SVBRDF over the pixels corresponding object surface area, the set of out-coming angles (reaching sensor) and a set of incoming light angles (exiting projector) be γ, which is constant over the scanning process. Instead of modifying cameras DR, we will change the light energy incoming to this area. We will send less light to the area with higher γ (high reflective materials), and more light to the surface part with lower γ (low reflective materials). This means, that the total dynamic range of our system will be the product of camera's and projector's DR. With this method, we will shrink the dynamic range of the scene, so it will fit into the camera's dynamic range. To modulate the projector's light energy, we develop a method for creating projector image weight map.

Moreover, we could send no light to strongly specular surfaces to reduce the effect of high frequency inter-reflections.

5.1 Projector image weight map

Projector image weight map consists of floats. It has a weight value for every projector pixel, which symbolizes the portion of energy to be used for each dedicated pixel. This weight map cannot be applied directly, because image value in projector image is not in linear relation with energy. The mapping between image value and energy is described by gamma curve (the power of 2.2 is often use in commercial digital projectors). In our solution, this mapping is accurately measured during calibration.

This mapping could be specified by bijection ***g***, which will map grayscale projector image values to values with linear response to the light energy. If the ***I*** is an arbitrary projector image and ***W*** is the weight map to be applied. We can compute final projected image as

$$I' = g^{-1}[g(I)*W]$$

The star symbol means multiplication per pixel.

5.2 Creating projector weight map

We can segment the overexposed and underexposed regions in camera image. To create the weight map, we need to know the correspondence mapping between camera and projector image space. Because, if we want to modify response of specific camera pixel, we have to choose correct projector pixel (pixels, because of aliasing), whose projection illuminates corresponding surface point. This correspondence is in general equivalent to surface reconstruction.

Point correspondence between the projector and the camera image space is dependent on the scene's geometry in general case. However, there exists one special position, in which the correspondence is constant. If we put camera in such a position, that camera and projector focal points are aligned.

5.3 Coaxial optical setup

Physical aligning of focal points is not possible, but we can simulate this with right optical solution, using beamsplitter and additional camera. The beamsplitter will split projector optical axis into 2. We will place new camera on optically created axis as it is shown in figure 5.

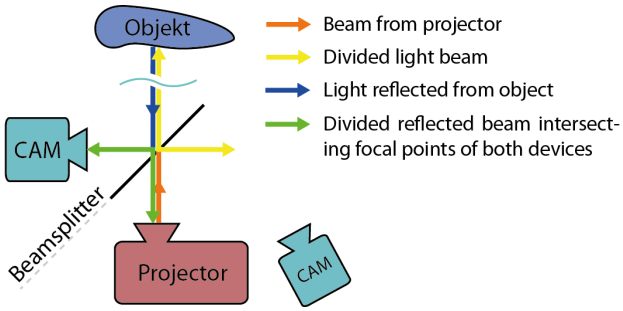


Figure 5: Our coaxial optical setup. Bottom camera is used for 3D reconstruction (Geometry camera). The additional camera is in the left

Light flow from the projector will be split into 2 parts. The reflected part needs to be eliminated by dark material, because it could possibly be reflected to the camera and create a noise. The transmitted part will illuminate the scene in the same way as in SMISS. The light reflected form the scene will hit beamsplitter. A part of it will lighten the projector, but a part of it will be reflected to the new camera.

If the added camera's focal point is placed to the projection of projector's focal point trough plane symmetry defined by beamsplitter plane of reflection, the correspondence between camera and projector image spaces will be constant and scene independent.

5.4 Image space correspondence

When we have our setting, we can calibrate the point correspondence between new camera and projector on theoretically arbitrary scene, but white diffuse flat table is recommended. For this purpose, we project Gray code patterns, both horizontal and vertical, so that every projector pixel has a unique binary vector code.

After this process, we have a mapping from camera image space to projector image space. This mapping is not injective in general, but we do not need inverse mapping. For every pixel of the projector, we count how many camera pixels point to it. These connections are then weighted accordingly. In practice, the camera resolution is often higher than the projector's. This means that more camera pixels could point to the same projector pixel. It is possible that some projector pixels become unaddressed, because of aliasing. For these points, the weight value will be set as an average of surrounding.

This calibration has to be done just once, because the correspondence does not depend on the scanned objects.

5.5 Iterative algorithm

This algorithm is executed before scanning. Images in algorithm are grayscale, and we use real numbers for value. In 8-bit image, the 255 correspond to 1 in our algorithm. For the camera, we have the default exposure, which is set according to the situation. The less responsive part of the camera image, which is directly lit by the projector (with full power), has to have a signal to noise ratio on sufficient level under the default exposure. This is because we can just scale down the power of projector lighting, so we need the default exposure, under which we can capture even the less responsive part of the image. We use conventional acquisition of HDR image for camera and the response values in this image are for the default exposure, so the image values could exceed 1.

```
W <- 1 //projector weight map
I <- 1 //Full white frame
Projector.Project(g'(g(I) * W))
P = Camera.CaptureHDR
  //Use conventional HDR   method
do
  W' <- 0 //new projector weight map
  foreach pixel in P do
    s <- 0.5 / pixel.value
      //scale power to the middle of dynamic
      //range
    W'[pixel.correspondence] <-
      W'[pixel.correspondence] +
      s * pixel.correspondence.weight
      //if there are for example 3
      //connections to pixel.correspondence,
      //the pixel.correspondnce.weight is
      //1/3
  //End foreach
  Compute weights for unaddressed points in
    W' from surrounding pixels
  Gauss(W')
    //Apply Gauss filter to W' with constant
    //sigma to compensate aliasing
  W <- W * W'
  Projector.Project(g'(g(I) * W))
  P = Camera.Capture
    //Use conventional method
until P do not contain overexposed points
  or maximum number of iterations was reached
```

After the algorithm ends, the final weight map is used for scanning program. In our prototype, we use 3 iterations as a maximum. Iterations in algorithm are used for adjusting defects of global HDR method. The result can be seen in figure 6.

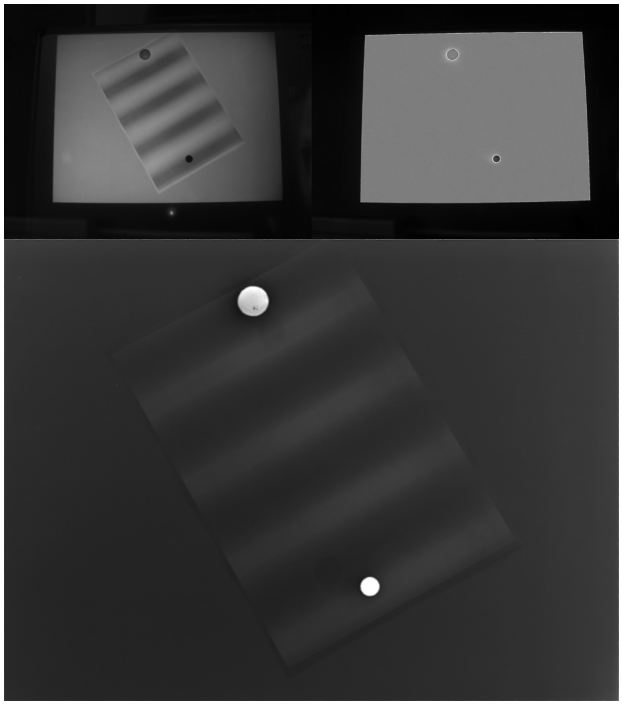


Figure 6: Top-left: camera image under constant white illumination. Top-right: camera image under projector map illumination. Bottom: calculated projector weight map

5.6 Weight map for geometry camera

We described how to create the projector weight map, using additional camera. However, we need the projector weight map benefit for geometry camera. If the whole scene is built from diffuse materials, than we can use the map directly. In reality, the image captured from geometry camera can be different, because specular part of light is view dependent.

For this purpose, we use technique for separation of diffuse and specular light component using polarization. We use linear polarizer to polarize the outcoming light from the projector. Let it have s-polarization. This light hits the scene. Diffuse like reflection is formed by light scattering in material microstructure. This type of reflection depolarizes the light polarization. On the other hand, the specular reflection maintains polarization, so the reflected light has the same angle of polarization, s-polarization. We have linear polarization filters on cameras, also. But the angle of polarization of the cameras filters is perpendicular to one in front of the projector. These filters transmit p-polarized light and

block s-polarized light. This means that the specular light component is blocked on polarization filters of the cameras. The diffuse part of light reflection, which is important for scanning, is un-polarized, so statistically, half of the energy goes through polarization filters (30 – 40 % in practice).

The technique also scales down the dynamic range of the scene, because it removes specular highlights. You can see the effect of the method on figure 7.

Figure 7: Scene without and with polarization filters

After application of this method, we could use our generated projector weight map for geometric camera.

6 Implementation

We implement the system presented in this paper from a scratch, both, hardware and software.

In our prototypes, we use conventional digital projector with combination of industrial grade digital cameras with linear light response. Whole setup is fixed on construction, so the relative position between components is constant.

The prototype of HDR SMISS was constructed independently of the system SMISS and the final fusion is planned for this year, because it requires new construction. Our co-axial prototype setting uses conventional beamsplitter, linear polarizers, dark surface to suppress reflected light, digital projector and camera. The whole setup could be found on figure 8. Our co-axial prototype setting has limited functionality, because projector light reflected from beamsplitter creates small systematic additive noise in HDR camera image.

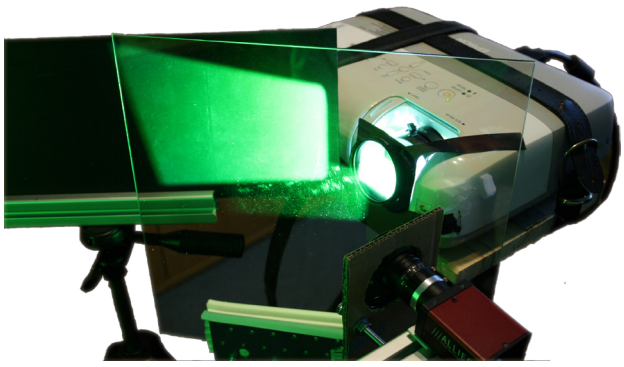


Figure 8: Our coaxial setup

In final setup, we plan to use a polarizing beamsplitter to optimize the optical flow efficiency, so only a minor part of projector polarized light will be reflected out of the system. The whole setup needs to be minimized, to get possible dust part on beamsplitter out of camera depth of field. We also consider setting, in which the beamsplitter is situated between projector modulating chip and its optics. In this setting, the additional camera chip shares the projector optics and everything is encapsulated in projector box.

Over the hardware, we create a seamless software solution, which encapsulate camera and projector control, calibration, data acquisition, scanning and processing. The system SMISS is designed to be a flexible platform, so addition of new methods is easy and straightforward.

Whole software was written in C++ (communication and processing) and C++/CLI (GUI), using OpenCV as a core library for image processing. The whole solution consists of more than 20 000 lines of code.

We implement easy to use semi automatic calibration process of intrinsic and extrinsic matrices for camera and projector, and also the calibration tool for calibrating projector response curve (gamma correction).

Our user interface is easy to use and can be controlled by conventional computer user without knowledge of 3D reconstruction, as the whole scanning process is automatic.

For visualization purposes, we use our OpenGL application, for which we develop fast quadratic interpolation method for filling of geometrical holes. We also construct our view dependent stereo vision setup for presentations.

7 Results

If we consider the noise level of high quality 8-bit grayscale camera to be *5*, the camera dynamic range would be *255 / 5*, what is approximately *50:1*. As we show in figure 4, the dynamic range of general scene could reach more than *400:1*. If we consider the response from direct reflections, the dynamic range of general scene could be far more than *10 000:1*. This is caused by varying material properties and different angle of incidence between projector light rays and the surface.

To capture scene with dynamic range of *400:1* with our camera and standard method, we would need $3 = log_2(400 / 50)$ times more frames. With our solution, we can build projector weight map with just 6 addition frames. Which means 36 frames in our full scanning setup, against *90 (30 * 3)*. So in this situation, our solution is *250* percent faster.

A conventional projector has dynamic range (contrast ration) approximately *1000:1*. With our technique, we could theoretically scan scenes with dynamic range of *50 000:1*. In practice, we can reach *10 000:1* because of projector noise.

Average accuracy of our system was measured as *0.5* millimeter, when we scan a flat ceramic table in distance of *800* millimeters.

We offer a few final renders of scanned models in figure 9. The triangulated mashes were scanned with SMISS setting from figure 2 (right) with digital projector and color camera aligned vertically inside of the box.

Figure 9: Rendered models captured with our 3D scanning solution

8 Discussion and future work

Our setting can be used to adaptively adjust scanning procedure. In [8], authors use different binary coding for materials affected by variable types of inter-reflection. The procedure consists of scanning object more times with different coding. In our future work, we plan to separate projector image regions according to affect of inter-reflection, and compose scene dependent optimal coding, so the scanning process will use just one set of new codes.

We are working on new sub-pixel precision method with use of defocused phase-shifting patterns (continuous pattern information). With pixel-precision geometry output from our system, we could compute the amount of defocus from Point Spread Function of the projector.

For now, our solution uses structured light emitted from point source to the scanned object. If the scanned object consisted of shiny (mirror like) surfaces, it would be impossible to scan the surface with our, or similar optical method. For this we want to extend the system for dual scanning method. The key idea is to illuminate the surface from globe around the object modulated by structured light. With this method, we want to measure surface normals, as an approximation of surface first derivation. We then want to use this dual information to relax the position of scanned geometry to satisfy scanned normals. This principle will lead to massive increase in scanning precision. An advantage of this method is also the possibility to scan glossy and mirror surfaces, so we will be able to approximate the geometry of object parts, which will consist of those materials.

With the similar setting, we plan to capture the approximation of BRDF function for individual surface point. With this knowledge, we will be able to produce physically correct photorealistic renders. This information can also be used for material description and segmentation, which could be important information for future 3D printers (description of building material for individual object parts).

This year, we plan to create construction and algorithms for full 3D scanning. The full 3D model will be merged from multiple scans from different position. For this purpose, we plan to track object position using Speckle Sense technology [17].

9 Conclusions

We present a novel approach to extend the addressed problem of the low dynamic range of scanning systems based on the structured light.

Our designed solution, with its co-axial setup is an easy to use, low cost platform for future improvements of the scanning process. Because of known camera – projector image correspondence, we could segment the scanning range to disjunctive parts, and process them individually. With different light power value, coding, or we can divide the scanning process in consequence of strong inter-reflections.

This paper is as an interim result of our research, but our SMISS prototype is fully functional and ready to use for basic shape and texture reconstruction, as we prove in Virtualny Svet (Virtual World) 2012 in Slovakia and Tire Technology Expo 2012 Exhibition in Germany, under the ME-Inspection company.

10 Acknowledgment

We want to thank the ME-Inspection company for hardware support and also for experience from industrial facilities. Greetings also belong to Tatra Banka E-Talent foundation for financial support through a grant.

References

[1] C. Nafis, *3D Scanners, Digitizers, and Software for making 3D Models and Measurements*. Cit. 10. Apr. 2011. Web page source: Simple3D: http://www.simple3d.com/

[2] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, & Scopigno, R., *A low cost 3D scanner based on structured light.* Computer Graphics Forum , 2001

[3] D. Lanman, G. Taubin, *Build your own 3D scanner: 3D photography for beginners.* ACM SIGGRAPH 2009 Courses , 2009

[4] T. Peng, S. K. Gupta, *Model and algorithms for point cloud construction using digital projection patterns.* ASME Journal of Computing and Information Science in Engineering, 2007

[5] S. S. Gorthi, P. Rastogi, *Fringe Projection Techniques: Whither we are?* Optics and Lasers in Engiering, 2010

[6] T. Chen, H-P. Lensch, C. Fuchs, H. P. Seidel, *Polarization and Phase-Shifting for 3D Scanning of Translucent Objects.* IEEE Conference on Computer Vision and Pattern Recognition, 2007

[7] S. Umeyama, G. Godin, *Separation of diffuse and specular components of surface reflection by use of polarization and statistical analysis of images.* Pattern Analysis and Machine Intelligence, IEEE Transactions, 2004

[8] M. Gupta, A. Agrawal, A. Veeraraghavan, S. G. Narasimhan, *Structured Light 3D Scanning in the Presence of Global Illumination.* Computer Vision and Pattern Recognition, 2011

[9] M. Holroyd, J. Lawrence, T. Zickler, *A Coaxial Optical Scanner for Synchronous Acquisition of 3D Geometry and Surface Reflectance.* ACM Transactions on Graphics, 2010

[10] M. O'Tool, K. N. Kutulakos, *Optical Computing for Fast Light Transport Analysis.* ACM SIGGRAPH Asia, 2010

[11] R. Hartley, A. Zisserman, *Multiple View Geometry.* Cambridge University Press, 2004

[12] Z. Zhengyou, *A flexible new technique for camera calibration.* IEEE Transactions on Pattern Analysis and Machine Intelligence , 2000

[13] S. K. Nayar, G. Krishnan, M. D. Grossberg, R. Raskar, *Fast separation of direct and global components of a scene using high frequency illumination.* ACM Transactions on Graphics , 2006

[14] E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, *High dynamic range imaging, 2nd Edition: Acquisition, Display, and Image-Based Lighting.* Morgan Kaufmann, 2010

[15] T. Kovačovský, *SMISS – Scalable Multifunctional Indoor Scanning System.* Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, 2010

[16] P. Debevec, J. Malik, *Recovering high dynamic range radiance maps from photographs.* Proceedings of 24th annual conference on Computer graphics and interactive techniques, 1997

[17] J. Zizka, A. Olwal, R. Raskar, *SpeckleSense: Fast, Precise, Low-cost and Compact Motion Sensing using Laser Speckle.* In Proceedings of UIST, 2011

Material Recognition: Bayesian Inference or SVMs?

Ishrat Badami*
Supervised by: Michael Weinmann,†
Reinhard Klein‡

Institute of Computer Graphics
University of Bonn

Abstract

Material recognition is an important subtask in computer vision. In this paper, we aim for the identification of material categories from a single image captured under unknown illumination and view conditions. Therefore, we use several features which cover various aspects of material appearance and perform supervised classification using Support Vector Machines. We demonstrate the feasibility of our approach by testing on the challenging Flickr Material Database. Based on this dataset, we also carry out a comparison to a previously published work [Liu et al., "Exploring Features in a Bayesian Framework for Material Recognition", CVPR 2010] which uses Bayesian inference and reaches a recognition rate of 44.6% on this dataset and represents the current state-of the-art. With our SVM approach we obtain 53.1% and hence, significantly outperform this approach.

Keywords: Material recognition, Texture classification, SVMs

1 Introduction

Understanding materials enables us to interact with the real world and influence our decisions in everyday life, e.g. where to drive a bike on a wet muddy road or whether a fabric in textile shop is smooth enough for cushion cover. These daily examples show the importance of material recognition for humans. In the fields of computer vision and computer graphics, one goal is to develop systems which can automatically perform this task. Identifying the respective material of object surfaces for instance allows to handle the object appropriately within a supply chain or to select the corresponding appearance properties for photo-realistic rendering.

For humans material recognition comes naturally. Since one can touch and feel the material surface if it is smooth or rough, hard or soft, take a look from different directions, from close or far distance and observe if it is shiny or dull. The key observation is that the visual appearance of a surface in an image depends on several different factors such as the illumination conditions, the geometric structure of the surface sample at several spatial scales, and the surface reflectance properties, often characterized by the bidirectional reflectance distribution function (BRDF) [23] and its variants [10, 15, 24].

Figure 1: We used Flickr Material Database [27]. This database captures a wide range of appearance of 10 different materials.

In order to capture such characteristics recent investigations [20] combine a large number of different low-level and mid-level features, which are commonly used in related areas such as object and texture recognition tasks, in a Bayesian framework. The authors demonstrated that their approach outperforms previous state-of-the-art methods [29] on a more challenging database [27].

Support Vector Machines (SVMs) [28, 7] have become popular for classification tasks, since they offer advantages such as, ease of generalization of the problem, its ability

*badami@informatik.uni-bonn.de
†mw@cs.uni-bonn.de
‡rk@cs.uni-bonn.de

to handle high-dimensional feature spaces and the absence of local minima [3]. As we want to focus on the comparison between the classification using the Bayesian approach in [20] and one based on SVMs, we combine the idea of using the image features from [20] within a SVM framework and exhaustively compare the achieved classification rates to the ones reported in [20]. For this, we also evaluate our approach on the challenging MIT Flickr Material Database [27]. We observe that with our system the recognition rate improves from 44.6% to 53.1%. We also evaluate our system on the KTH_TIPS2 dataset [4].

The rest of this paper is organized as follows: In Section 2, we describe the present state of methods used in the area. In Section 3, we introduce feature pools used for classification. In Section 4, we explain the support vector machine classification model in the context of material recognition. Finally, in Section 5, we examine our system on the Flickr Material Database and discuss the results. In Section 6, we conclude.

2 Previous work

Learning high-level material categories such as foliage, stone or metal is related to object and texture classification but differs in several aspects. Several approaches address material recognition by focusing on purely texture based image features.

Texture has been defined in terms of dimensions like periodicity, orientedness and randomness [21]. A recent work on 3D textons [19] addresses material recognition using multiple images of varying viewpoint and lightning conditions. Cula and Dana [8] adapted the method of [19] to 2D textons where each histogram is obtained from a single image in the training set. For their evaluation they used the CUReT database [10] consisting of images of 61 different texture samples under 205 different viewing and illumination conditions. A high classification rate of more than 95% is reported in [29] with 2D textons on the CUReT dataset using the NN classifier. In [4] it is shown that the SVM based classifier achieves 98.5% accuracy on the KTH_TIPS2 [4] database consisting of 11 material categories with 4 texture samples in each category photographed under various conditions. Although texture is a characterizing feature cue it is not sufficient for representing material properties completely. It might happen that an object made of different materials has a similar or even the same texture.

Although information about objects can lead us to the right guess concerning the material from which it is made of, sometimes it is really misleading. For example a cup can be made of plastic, metal or glass. In case of an artistic cup it can be carved out of wood or stone as well. This demonstrates the difference between material recognition and object recognition.

The appearance of a material in an image highly depends on the environment illumination and surface reflectance properties described by the BRDF. Material recognition might be trivial in case of a known BRDF. But it is very difficult to estimate the BRDF of the material from a single image without simplifying assumptions [11].

Moreover, the appropriate choice regarding the classification method is also an influencing aspect. While a few approaches make use of a Nearest Neighbor (NN) classifier (e.g. [29]), the methods we consider as state of the art rely on a Bayesian framework [20] or SVMs [4]. SVMs have been proven to consistently achieve good performance in complex real-world problems such as text [16, 12] and image classification [6] and bioinformatics [30] and biosequence [2] analysis. This motivates us to involve SVMs as classifier. However, most of the methods that address material classification are evaluated on datasets such as [10] that are not well-suited for this task as they do not contain the large variations in appearance which occur in real-world scenes and for this, classification rates usually are very high. Hence, the real performance differences cannot be seen in a reliable way. The reason for this is that they have been acquired using a controlled setup. In contrast, [20] show the performance of a Bayesian approach on the significantly more challenging MIT Flickr Material Database [27]. However, they only compare their method against the NN classifier and not against SVMs. We believe that a real comparison between different material classification approaches has to be carried out on such a challenging dataset. Hence, we compare our SVM based technique to the method described in [20].

3 Feature Extraction

For the development of a reliable image based material recognition system it is important to consider image features which are representative and discriminative. However, it seems to be impossible to capture the variety of material characteristics in a single feature descriptor, as commonly used descriptors usually are restricted to a certain material property such as color, texture or reflectance behavior. As the different material properties are not equally descriptive for different material classes, a variety of features have to be considered in order to derive information about the different materials of an object. If the object appears shiny one might think that it is made of glass or metal whereas an object surface covered by minute fibers appear rough and together with the underlying weave pattern leads to a specific textured representation within an image. Wood is recognized usually with its brown color. In order to take several characteristic material properties into account, we follow the idea of [20] and use a pool of features which are covering different aspects of appearance. In general, for a fixed camera and object position, the image can be determined by 1) BRDF, 2) surface structure, 3) color, 4) object shape and 5) environment illumination. As we want to obtain hints on the performance of the SVM classifier in comparison to the Bayesian frame-

work proposed in [20] we extract the same image features.

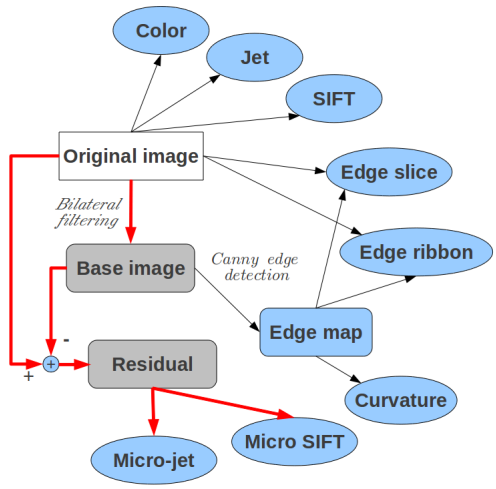


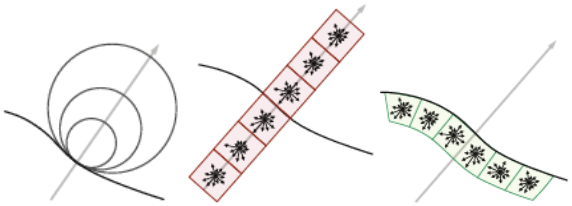
Figure 2: Features used in the classification [20].

Color is an important cue for recognizing materials. For example foliage is green, wood is usually brown and stone has less saturated color whereas fabric, plastic and paper have saturated color. To capture local color information, we store the RGB values in a local 3×3 neighborhood and concatenate them to a vector of dimension 27 as in [20].

Furthermore, every material has a peculiar texture. Wood has a ringing pattern, whereas fabric has a weaving pattern. Like [20] we use two sets of features to characterize texture. The first feature is SIFT [22] which is commonly used as a texture feature and also serves for tasks such as object and scene recognition. The second set of features is responses of an image through a set of Gaussian filters of different scale and orientation, also known as Jet [17]. We consider 2 Gaussian derivative filters at 3 scales and 6 orientations, i.e. $2 \times 3 \times 6 = 36$ rotational variant filters, and 8 Laplacian of Gaussian and 4 Gaussian filters, i.e. 8+4 = 12 rotational invariant filters. We combine all the filter responses in a single vector of size 48.

In addition, it is important to capture features not only on a meso-level but also on a micro-level. The human visual perception system can impressively abstract minute details of texture, e.g. smoothness of metal and glass surfaces, grains in paper and stone, the fibers of the fabric and crinkles in leather. For extracting micro texture of an image we follow the idea in [1] where the image is smoothed by a bilateral filter [13] and the residual is obtained by subtracting the smoothed image from the original image. The obtained residual image is used for further analysis as it reveals the information of texture on a finer scale. We derive descriptors that capture such micro details by computing SIFT and Jet over the residual image. For micro-texture, the Jet filter bank is evaluated on the same set of orientations but on a different set of scales in comparison to the Jet applied to the original image.

Materials can be molded to any arbitrary shape to create different objects, but still the outline shape of an object and its material category are often related, for e.g. fabric and glass have a curvy structure whereas metal, wood, stone can have straight edges and sharp corners. The edges and corners can be acquired from edge maps. We extract such edge maps by applying the canny edge detectors [14] to the base image. Furthermore, we only consider edges having a certain minimum length. Corresponding examples are shown in Figure 3. The curvature along these edges can be used to represent the orientation of the outline shape of a certain material, we calculate this specific descriptor by sampling the edges at three different scales (see Figure 4(a)), which results in a 3D-vector. As we are interested in a dense sampling, we calculate such a descriptor for every second pixel along the edges.



(a) Curvature (b) Edge-Slice(HOG) (c) Edge-Ribbon(HOG)

Figure 4: Curvature is calculated over three different scale, Edge-Slice and Edge-Ribbon are calculated in 6 cells [20] at edges.

Furthermore, reflectance behavior is also an important cue for classifying material categories. Water and glass are translucent, metal is shiny, wood and stone are dull and opaque. Such properties can be observed in form of distinctive intensity changes at the edges in an image. We follow [20] in computing histograms of oriented gradients (HOG) [9] in the vicinity of the edges. More precisely, we first select a slice of a certain width along the normal direction of the edge and compute the gradient at all of the pixels inside the slice. In order to calculate HOG, the slice is divided into 6 cells where the gradient orientation is quantized into 12 bins. We combine the histogram of all 6 cells in a vector of length 72, which will be referred as Edge-Slice [20]. In addition, we use the same method and employ a slice along the tangent direction of the edge in order to obtain the Edge-Ribbon feature [20](see Figure 4(b) and 4(c)).

So far, we described all the features which we use to characterize material appearance due to different properties. Figure 2 shows a flowchart how the features are generated. Among these features color, SIFT and Jet are low level features and can be calculated directly from the image. In contrast, curvature, Edge-Slice, Edge-Ribbon, micro-SIFT and micro-Jet are mid level features which depend on the edge map and the base image respectively (see Figure 3). In order to capture the relevant information appropriately, we calculate color, SIFT, micro-SIFT, Jet and micro-Jet on a evenly sampled grid in the image

(a) Original (b) Base image(bilateral filter) (c) Edge map(Canny) (d) Residual (a)-(b)

Figure 3: **Example images of how features are calculated in our system**. From top to bottom the rows show examples for fabric, glass and wood. On image (a) we apply bilateral filtering [13] to obtain the base image (b). We run the Canny edge detector [14] on the base image and compute edge maps. Curvature, Edge-Slice and Edge-Ribbon are extracted as a feature from the edge map. Subtracting (b) from (a), we get the residual image (d) that depicts micro structures of the materials which are captured by micro-SIFT and micro-Jet features.

areas where the material has been annotated. The remaining features are sampled at every second pixel along the edges.

4 Classification with SVMs

Once the features have been calculated we want to build a robust material recognition system. For this, we first apply a quantization of the features to form characteristics clusters whose centers are denoted as visual words. In the next step, we use SVMs for material classification based on the given inputs. In the following, we will describe these steps in more detail.

4.1 Feature quantization and visual words

Before we start classifying our features we need to group alike features to reduce the massive data into few representative visual words. From training images we estimate visual words which we can expect being present in the test data as well. We use k-means clustering for the quantization. After quantizing individual features into k visual words, the distribution of visual words per image is calculated for all of the different features by assigning each pixel in the image the nearest visual word index and calculating the histogram over the frequency of the visual words. Figure 5 shows some clusters for different feature types.

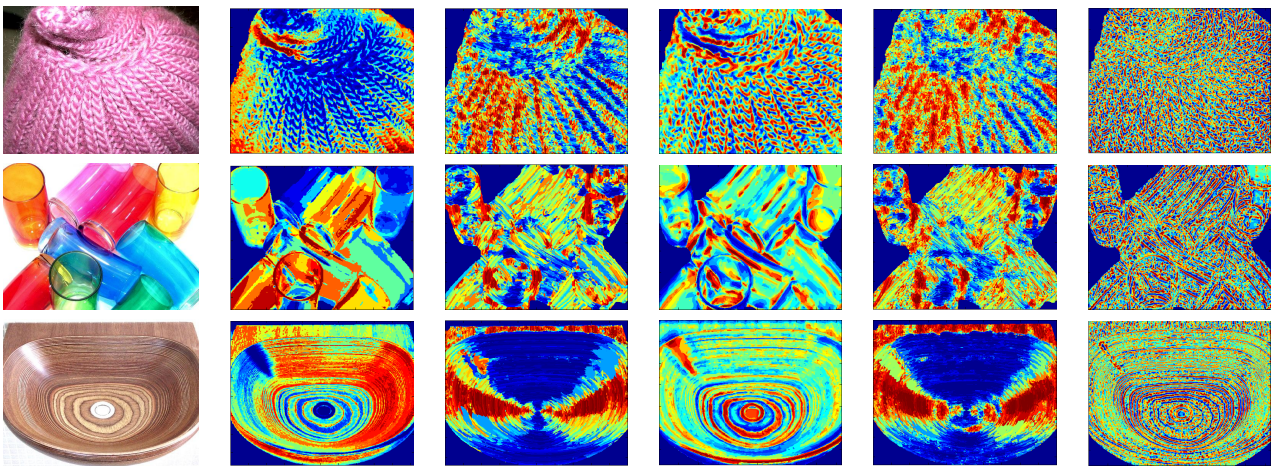
To generate a common visual word dictionary, for all the different types of features, suppose there are m features in the feature pool (e.g. color, SIFT, Jet) and m corresponding dictionaries $\{D_i\}_{i=1}^{m}$. Each dictionary has V_i codewords (e.g color has 150, SIFT has 250), i.e. $|D_i| = V_i$. Since the features are quantized separately the words generated by the i-th feature are $\{w_1^{(i)},.....,w_{N_i}^{(i)}\}$, where, $w_j^{(i)}$ is an index representing j-th cluster center of i-th feature, $w_j^{(i)} \in \{1,2,...V_i\}$ and N_i is the number of words. In order to combine two features, the corresponding dictionaries are simply put together. For example, a document of m sets of words

$$\{w_1^{(1)},....,w_{N_1}^{(1)}\},\{w_1^{(2)},....,w_{N_2}^{(2)}\},.....,\{w_1^{(m)},....,w_{N_m}^{(m)}\} \quad (1)$$

can be combined to one set

$$\{w_1^{(1)},....,w_{N_1}^{(1)},w_1^{(2)}+V_1,....,w_{N_2}^{(2)}+V_1,.....,w_1^{(m)}+\sum_{i=1}^{m-1}V_i,....,w_{N_m}^{(m)}+\sum_{i=1}^{m-1}V_i\} \quad (2)$$

with a joint dictionary $D = \cup_i D_i, |D| = \sum_{i=1}^{m} V_i$. In case of combining color ($V_1 = 150$) and SIFT ($V_2 = 250$) the



(a) Original (b) Color (c) SIFT (d) Jet (e) micro-SIFT (f) micro-Jet

Figure 5: **Visualization of quantized features**. After finding k cluster centroids of individual features, each pixel is assigned an index of the closest visual word. In order to visualize how the cluster centers are distributed in an image, the corresponding indices are color coded by RGB values. Same colors indicate that feature vectors corresponding to the pixels lie in the same cluster.

first 150 entries of the dictionary (of size $V_1 + V_2 = 400$) are codewords of color and the next 250 entries represent codewords of SIFT. This way we can reduce the multi-dictionary problem to a single dictionary problem.

4.2 Support Vector Machines

Being robust to noise [25] and being capable to generalize in case of a small training set [26], SVMs have become a popular and commonly used classifier for recognition tasks. A single SVM constructs a hyperplane or set of hyperplanes in a high-dimensional space and allows to distinguish between two linearly separable sets of samples. In order to deal with our multiple classes given in the used data sets, there is a need for using an SVM formulation which is capable of dealing with more than two classes. This can be achieved by either using several pairwise classifiers arranged in trees [18], where each of the nodes represents an SVM, or by using an one-vs-others approach, where multiple SVMs are trained and each of them separates a single class from all remaining ones. We follow the first strategy and use the implementation in [5].

As SVMs perform a supervised learning, objects with known class labels are used as samples for the training phase. We use the training data $\{\mathbf{x}_i, yi\}$ with $i = 1,\ldots,l$ where $\mathbf{x}_i$ represents the histogram of visual words per image for a single feature type or a feature combination and y_i describes the corresponding material category in $\{fabric, foliage, \ldots, water, wood\}$, i.e. $y_i \in \{1,2,\ldots,10\}$. As kernel function, we use the Gaussian RBF kernel

$$K(x_i,x_j) = e^{\frac{-\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}. \quad (3)$$

5 Results

We run our system on two data sets namely MIT Flickr Material Database [27] and KTH_TIPS2 database [4].

First, we tested our system on the KTH_TIPS2 database. In this database there are 11 different materials namely *Crumpled aluminum foil, Cork, Wool, Lettuce leaf, Corduroy, Linen, Cotton, Brown bread, White bread, Wood, Cracker*. There are 44 different material samples present in the database in total. For each sample, images are taken at 9 scales, 3 poses and 4 different illumination conditions, hence there are $44 \times 9 \times 3 \times 4 = 4752$ images in this database.

We extract the same features and feature combinations as in [20] for this database and the results are plotted in Figure 8. The highest recognition rate is achieved by 99.4%.

As mentioned before, we do not consider this dataset to be challenging enough to derive statements on performance differences within complex scenes. For this, we consider the MIT Flickr Material Database [27], where there are 10 material categories namely *fabric, foliage, glass, leather, metal, plastic, paper, stone, wood, water*. Each category contains 100 images. 50 images show close-up views of the materials and 50 show an object made of the corresponding material. This dataset contains also annotations where this specific material is located in the image domain. Only pixels inside these areas are considered for feature calculation. For training, we randomly choose 50 images per category and test the system on the rest. For reliable results, we take 25 images showing close-up views and 25 images showing the full object made of the material. In addition, the training/testing process is repeated 5 times for different, randomly chosen training sets and the classification rates are averaged.

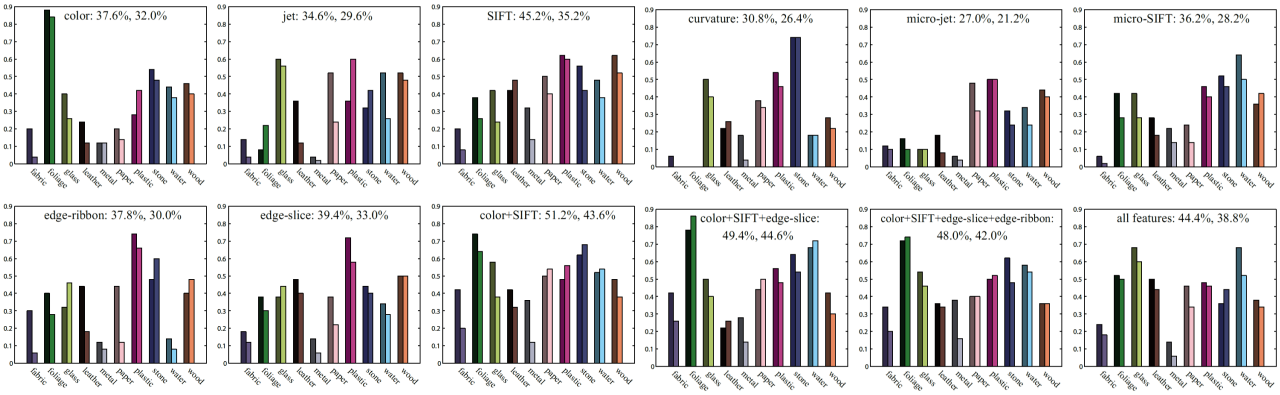


Figure 6: The per-class recognition rate (both training and test) with different sets of features for the MIT Flickr Material Database [28], using the classification approach proposed in [20]. In each plot, the left, darker bar means training, the right and the lighter bar means test. The two numbers right after the feature set label denote the recognition rate on the entire training set and on the entire testing set.

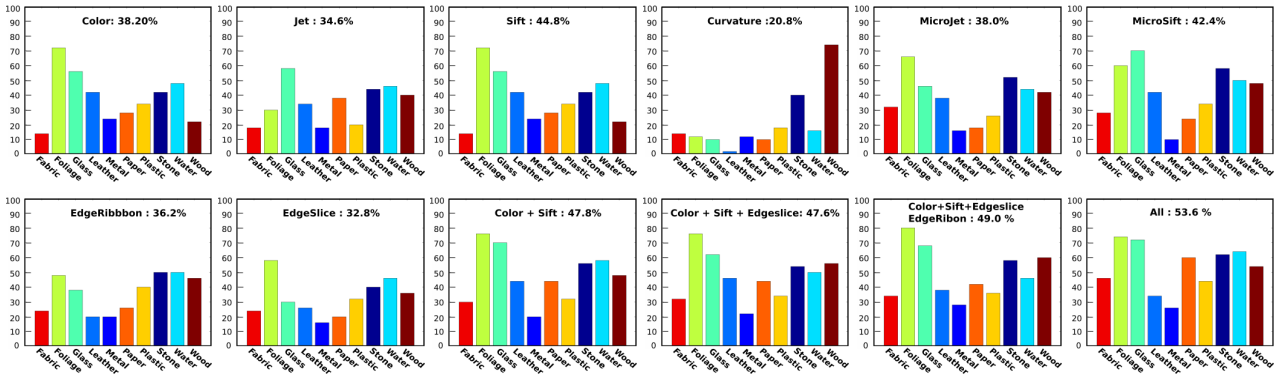


Figure 7: The recognition rate of a randomly chosen split of training-testing shown for each single feature and some of the feature combinations. Only the test rate per material category is mentioned, since the training rate with learned parameters is very high and hence not of our interest. The highest recognition rate (53.6%) is achieved when all the features are used.

We extract the features color, SIFT, Jet, micro-SIFT and micro-Jet on every 5-th pixel inside the given mask where the material is present. Edge-Slice, Edge-Ribbon and curvature are calculated on every second pixel on the edges in the edge map. After calculating the features we perform k-means clustering separately for each feature. We use exactly the same number of clusters for each feature type (150 for color, 250 for SIFT, 200 for Jet, 250 for micro-SIFT, 200 for micro-Jet, 100 for curvature, 200 for Edge-Slice, 200 for Edge-Ribbon).

After forming the dictionary for each feature we learn visual word histograms based on the training images using SVMs. First we train and test with single features and then we combine the features as in [20]. We observe that our best performing single averaged feature (SIFT = 42.2%) has a recognition rate which is very close to the best performance (44.6%) stated in [20]. Among all features and feature combinations, in every trail we achieve the best rates (53.2%, 53.6%, 53.8%, 53.6%, 51.6%) when all the features are combined. A comparison of average recognition rates of individual features and feature combinations is tabulated in Table 1. It can be seen that for most of the features there is an improvement in recognition accuracy of about 4 to 5 %. For the individual features, micro-SIFT and micro-Jet and the combination of all the features there is a significant improvement in the classification accuracy. The confusion matrix in Figure 9 shows the accuracy of the classification of individual material categories. The rates of misclassification are also shown.

6 Conclusions

In this paper, we addressed the problem of material recognition using various image features in combination with a SVM framework and compared it to the Bayesian approach proposed in [20]. The recognition rate achieved by our system is 53.1% in average on the MIT Flickr Material Database and 99.4% on the KTH_TIPS2 database. The reason for the huge difference in recognition rate between the two datasets is due to the larger intra-class variations

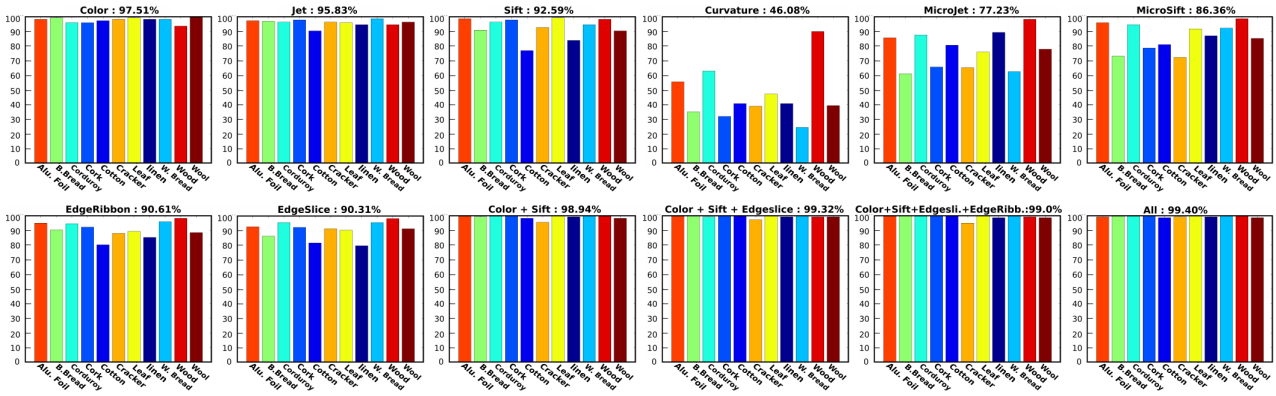


Figure 8: The recognition rate of a randomly chosen split of training-testing for KTH_TIPS2 database is shown for each single feature and some of the feature combinations. Only the test rate per material category is mentioned, since the training rate with learned parameters is nearly 100% for this dataset. The highest recognition rate (99.4%) is achieved when all the features are used.

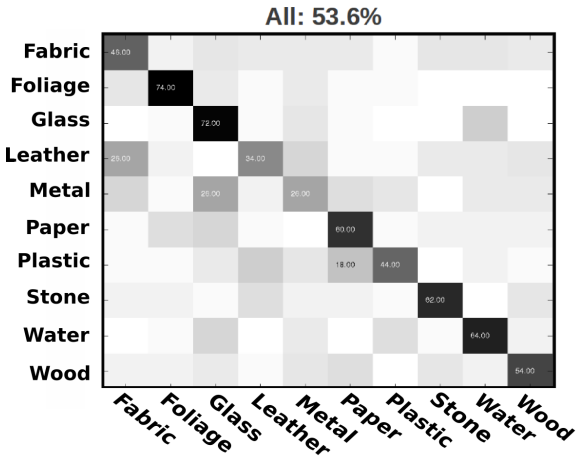


Figure 9: Confusion matrix for the Flickr Material Database. Diagonal entries shows the percentage with which each category is recognized. Rates are color coded using gray scale values (black = 100% and white = 0%). Each row sums to 100%.

in the Flickr Material Database. In contrast, KTH_TIPS2 comprises images of the same material taken in different view and lighting conditions in the same material category. We showed that with the same pool of features as given in [20], SVM classifies materials with a higher rate in comparison to the Bayesian approach of [20]. We have also analyzed the contribution of each feature in our system to the performance gain. Future developments should consider exploring different and better characteristic features for materials. Improvements by integration of different classification techniques can also be investigated.

Feature	**Ce Liu et.al**	**Our model (avg. of 5 iterations)**
Color	32.6 %	37.6%
Jet	29.6 %	34.0%
SIFT	35.2 %	42.2%
Curvature	26.4 %	21.6%
Micro-Jet	21.2 %	36.5%
Micro-SIFT	28.2 %	42.0%
Edge-Ribbon	30.0 %	36%
Edge-Slice	33.0 %	34.6%
Color+SIFT	43.6 %	48.6%
Color+SIFT+Edge-Slice	**44.6** %	49.1%
Color+SIFT+Edge-Slice +Edge-Ribbon	42.0 %	49%
All	38.8 %	**53.1**%

Table 1: Performance comparison between [20] and our system.

References

[1] S. Bae, S. Paris, and F. Durand. Two-scale tone management for photographic look. *ACM Trans. Graph.*, 25:637–645, July 2006.

[2] G. Bejerano. *Automata learning and stochastic modeling for biosequence analysis*. 2003.

[3] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[4] B. Caputo, E. Hayman, M. Fritz, and J. Eklundh. Classifying materials in the real world. *Image Vision Comput.*, 28:150–163, January 2010.

[5] C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[6] O. Chapelle, P. Haffner, and V. Vapnik. Support vector machines for histogram-based image classification. *IEEE Transactions on Neural Networks*, 10(5):1055–1064, 1999.

[7] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[8] O. G. Cula and K. J. Dana. Compact Representation of Bidirectional Texture Functions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1041–1047, 2001.

[9] N. Dalal and B. Triggs. Object detection using histograms of oriented gradients. *European Conference on Computer Vision Workshop on Pascal VOC*, 06, 2006.

[10] K. J. Dana, B. van Ginneken, S. K. Nayar, and J. J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Trans. Graph.*, 18:1–34, January 1999.

[11] P. Debevec, T. Hawkins, C. Tchou, H.P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 145–156, 2000.

[12] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, CIKM '98, pages 148–155, 1998.

[13] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 257–266, New York, NY, USA, 2002. ACM.

[14] Canny J. A computational approach to edge detection. *IEEE Trans. Pattern Analysis Machine Intelligence*, 8(6):679–698, 1986.

[15] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan. A practical model for subsurface light transport. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 511–518, 2001.

[16] T. Joachims. Text categorization with support vector machines : Learning with many relevant features. *Machine Learning ECML98*, 1398(23):2–7, 1998.

[17] J. J. Koenderink and A. van Doorn. Representation of local geometry in the visual system. *Biological Cybernetics*, 55:367–375, 1987.

[18] Ulrich H.-G. Kreßel. *Pairwise classification and support vector machines*, pages 255–268. MIT Press, Cambridge, MA, USA, 1999.

[19] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Comput. Vision*, 43:29–44, June 2001.

[20] C. Liu, L. Sharan, E. H. Adelson, and R. Rosenholtz. In *CVPR*.

[21] F. Liu and R.W. Picard. Periodicity, directionality, and randomness: Wold features for image modeling and retrieval. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 18:722–733, 1996.

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

[23] F. E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied Optics*, 4(7):767–775, 1965.

[24] S. C. Pont and J. J. Koenderink. Bidirectional texture contrast function. *Int. J. Comput. Vision*, 62:17–34, April 2005.

[25] M. Pontil and A. Verri. Properties of Support Vector Machines. Technical Report AIM-1612, 1997.

[26] D. Roobaert, M. Zillich, and J. Eklundh. A Pure Learning Approach to Background-Invariant Object Recognition Using Pedagogical Support Vector Learning. In *Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Kauai, Hawaii*, pages 351–357, 2001.

[27] L. Sharan, R. Rosenholtz, and E. H. Adelson. Material perception: What can you see in a brief glance? [abstract], 2009.

[28] V. Vapnik. *The nature of statistical learning theory*. Springer, New York, 1995.

[29] M. Varma and A. Zisserman. A statistical approach to material classification using image patch exemplars. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31:2032–2047, November 2009.

[30] J. Wang. Application of Support Vector Machines in Bioinformatics. Master's thesis, 2002.

Natural Phenomena & Perception

Real-time Lighting Effects using Deferred Shading

Michal Ferko*
Supervised by: Michal Valient†

Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava / Slovakia

Abstract

Rendering realistic objects at interactive frame rates is a necessary goal for many of today's applications, especially computer games. However, most rendering engines used in these games induce certain limitations regarding moving of objects or the amount of lights used. We present a rendering system that helps overcome these limitations while the system is still able to render complex scenes at 60 FPS. Our system uses Deferred Shading with Shadow Mapping for a more efficient way to synthesize lighting coupled with Screen-Space Ambient Occlusion to fine-tune the final shading. We also provide a way to render transparent objects efficiently without encumbering the CPU.

Keywords: Real-time Rendering, Deferred Shading, High-dynamic range rendering, Tone-mapping, Order-Independent Transparency, Ambient Occlusion, Screen-Space Ambient Occlusion, Stencil Routed A-Buffer

1 Introduction

Our rendering engine is based on concept of Deferred Shading [3], which avoids shading occluded pixels and by postponing the lighting evaluation allows one pixel to be affected by hundreds of lights.

Our system uses HDR rendering coupled with the tone-mapping operator by Reinhard et al. [11] and Bloom, Shadow Mapping [16] to provide hard-edged shadows, Screen Space Ambient Occlusion to simulate indirect lighting and Stencil Routed A-Buffer [8] to render transparent objects. All these techniques allow easy integration into a deferred renderer while providing much more realistic display of scenes.

The main contribution of this paper is a complete rendering pipeline incorporating these well-known techniques. Our aim is to determine in which order these techniques should be applied to avoid incorrect artifacts and how to maintain reasonable quality while allowing real-time display even on older hardware.

*michalferko1@gmail.com
†michal.valient@guerilla-games.com

We are targeting OpenGL 3 capable hardware, because we require the framebuffer object features as well as multiple render targets.

2 Related Work

There are many implementations of Deferred Shading and this concept has been widely used in modern games [15] [12] [5], coupled with techniques used in our paper as well as certain other.

Deferred Shading does not directly allow rendering of transparent objects and therefore, we need to use a different method to render transparent objects. There are several approaches to hardware-accelerated rendering of transparent objects without the need to sort geometry. This group of algorithms is referred to as Order-Independent Transparency.

An older approach is Depth Peeling [7] [4], which requires N scene rendering passes to capture N layers of transparent geometry. Dual Depth Peeling [1] improves the algorithm by capturing two layers of geometry in one pass. However, the objects still need to be rendered multiple times and the performance is still unacceptable for large scenes. Once the layers are captured, a final fullscreen pass blends them together.

A newer approach, Stencil Routed A-Buffer [10], allows capturing of up to 32 layers during one rendering pass thanks to multisample render targets on OpenGL 3 hardware. An additional pass for sorting the values is used. This approach is part of our system.

With OpenGL 4 hardware, it is possible to actually have per-pixel linked lists [13] and thus generate an arbitrary number of layers and sort these samples afterwards in a fullscreen pass. This approach is very similar to Stencil Routed A-Buffer except for the way the samples are stored. We did not use this approach due to the lack of OpenGL 3 support.

To further improve the visual quality of rendered images, we include standard Shadow Mapping [16] for shadow-casting lights and real-time Ambient Occlusion. There has been much research done regarding real-time Ambient Occlusion. In [2], the authors convert polygonal meshes into disks, for which the occlusion computation is simplified and allows dynamic rendering. In [6], the au-

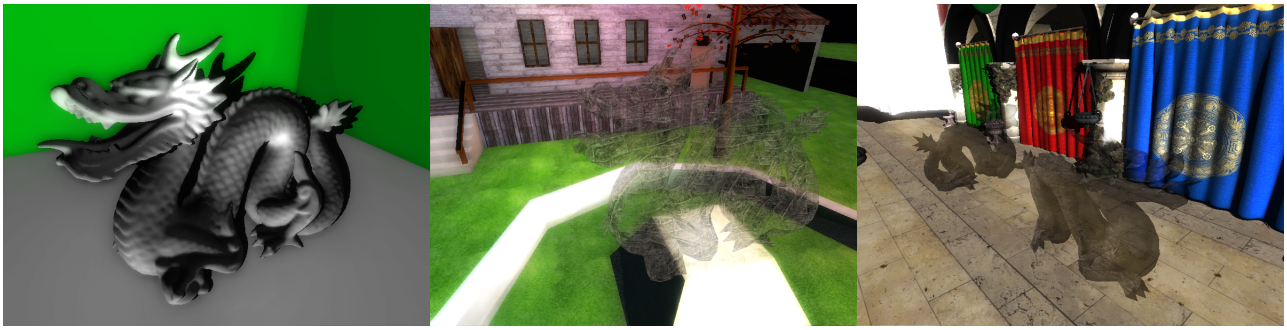


Figure 1: Images of test scenes rendered with our system at interactive frame rates. Dragon scene (Left), House scene (Middle) and Sponza scene (Right)

thors propose a method to generate fields around objects in pre-processing. As long as the objects are not deformed, the fields do not need to be recomputed. It thus allows real-time estimation of occlusion. A similar technique [8] performs a slightly different field generation in the geometry shader and allows for fully-dynamic ambient occlusion even on deformable meshes.

Our work is based on Screen-Space Ambient Occlusion [9] which uses the scene depth information captured during the first stage of deferred shading to approximate scene geometry and thus compute occlusion. The choice was made mainly due to the fact that the previous methods are scene-dependent and perform slower than SSAO when the scene contains hundreds of thousands of triangles. SSAO's performance depends only on the number of samples taken and the screen resolution, being totally independent from the actual scene.

3 Deferred Shading

Deferred Shading [3] is an alternative to Forward Shading, the traditional rendering where a fragment shader accesses all light information at once and outputs the final light contribution directly into the window's framebuffer.

The main idea of Deferred Shading is separation of the lighting calculations from the scene rendering pass (or the geometry pass). During this pass, material and object properties (usually albedo, depth, normal and specular power) are stored into a geometry buffer (G-Buffer).

When compared to forward rendering or multi-pass rendering, the scene is rendered only once and only the fragments that are visible are shaded. No shading needs to be evaluated for objects that are not affected by a certain light (the object is outside of the light volume - part of the scene that is affected by the light).

During the consecutive lighting pass, the light volumes are rendered (cones for spot lights and spheres for point lights) and during the fragment shader execution, the G-Buffer data is read and used to synthesize lighting. The light shapes are rendered with additive blending thanks to the additive nature of light. The results can be displayed on the screen, but usually more post-processing steps are executed after this pass and the results should instead be rendered into a texture - the lighting buffer (L-Buffer).

When rendering light shapes, we use front-face culling to avoid problems when the camera is inside a light volume. Furthermore, for every pixel getting rendered, it is needed to reconstruct the eye-space position corresponding to the current pixel position and the depth stored in the G-Buffer. For this position, we calculate whether it actually is inside the light volume, since we might be rendering nearby light volumes while the G-Buffer contains information about distant objects unaffected by the light.

Deferred Shading mainly outperforms Forward Shading when there are many lights that do not cover large portions of the screen when being rendered. Many directional lights (which affect the entire screen) pose a problem for Deferred Shading, because we do extra work when compared to Forward Shading. Therefore, some implementations evaluate directional lights using Forward Shading [5] and all other lights using Deferred Shading.

In a typical outdoor scene there is usually one directional light representing the sun and in indoor scenes, directional lights are avoided altogether. Our system calculates directional lights with forward shading during the G-Buffer generation phase. Due to this fact, only a fixed amount of directional lights can be forward shaded, for more lights the system would have to switch to deferred shading for the additional lights.

3.1 HDR and Tone-mapping

Having evaluated lighting in a texture adds direct support for HDR rendering. Performing tone-mapping during the L-Buffer generation phase is not possible, since the results get additively blended and we cannot guarantee that a pixel will not be affected by a large number of lights, resulting in a sum of many tone-mapped values which result in a luminance value higher than 1.

Our system is open to many tone-mapping operators. Currently, we are using the global part of the operator by Reinhard et al. [11]. As an input into the tone-mapping stage, we have the L-Buffer which contains 16-bit floating

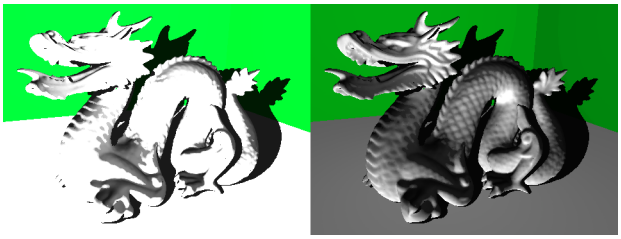


Figure 2: Real-time Reinhard's tonemapping. HDR image before applying tonemapping (Left) and after applying tonemapping (Right).

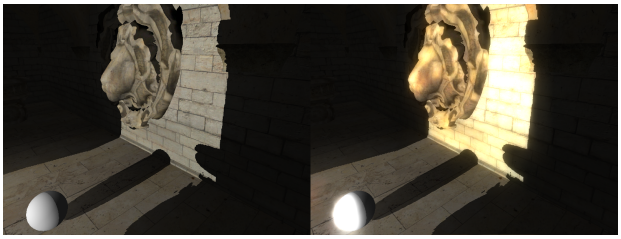


Figure 3: A scene with Bloom disabled (Left) and enabled (Right). Notice the light leaking into the shadow.

point RGB values.

Reinhard's operator analyzes the whole input image and tries to estimate whether it is light or dark. Based on the average luminance of the image L, the luminances are tonemapped into a valid range.

We perform a traditional GPU accelerated calculation of the average luminance of the L-Buffer by downscaling the image up to a 1x1 texture (using mipmap generation) and the value of this one pixel is the average image luminance.

3.2 Bloom

When looking at an object that is occluding a part of a bright light source, the light rays appear to "bend" at the object's edge and a fringe is visible on the occluding object. This effect is called *Bloom*, and it is usually coupled with HDR lighting since only very bright light sources produce such an effect. A scene with and without Bloom is shown in Figure 3.

Our implementation is tightly coupled with the tonemapping operator we use. After calculating the average luminance L of the image, we subtract L from the intensity of pixels in the L-Buffer (we ignore low-luminance pixels) and store these as a tresholded image. Pixels with luminance above average are marked in this image and we perform a blur based on how much brighter the pixels are when compared to average luminance.

Instead of performing a gaussian blur of different kernel size in each pixel, we chose to generate mipmaps for the tresholded image and sum values from multiple mipmaps, while alpha blending into the final image based on resulting pixel luminance.

3.3 Shadow Mapping

Incorporating Shadow Mapping into a Deferred Renderer is straightforward and allows for hard-edged shadow without much effort.

Shadow Mapping [16] is a fully GPU accelerated method for rendering real-time shadows. The entire scene is rendered one additional time (from the light's point of view in the light's direction) for every shadow-casting light, storing the depth values generated during this rendering.

Afterwards, when evaluating the lighting equation, the world position of the current surface point is projected (using the same projection as was used during the shadow map generation phase and remapping from $[-1,1]^3$ range into $[0,1]^3$) which gives us the (x,y) coordinates in the shadow map and a depth value z. The fragment shader reads the depth value d at position (x,y) in the shadow map and compares it to z. If $z = d$, the point with depth z is the closest point to the light source and therefore is not shadowed. If $z > d$, this point is behind a point with depth d and is therefore not directly lit by the light.

Due to floating-point precision errors, a small offset needs to be added to all values stored in the shadow map, otherwise non-realistic self-shadowing artifacts occur.

Integrating Shadow Mapping into a Deferred Renderer is simple. The scene is rendered one additional time for every shadow-casting light and thanks to the fact that we only access one light at a time in the fragment shader, we can reuse one texture for multiple lights. Our system does this by flip-flopping between shadow map generation and L-Buffer generation. We first generate the shadow map for the first light, then render the light's shape into the L-Buffer while accessing the map. We then clear the shadow map, render from the second light's point of view (into the same shadow map), and render the second light's shape into the L-Buffer. For lots of shadow-casting lights, this is a necessary approach due to the fact that we already took up a lot of memory with the G-Buffer (and the L-Buffer).

4 Transparent Objects

The Deferred Shading approach does not support rendering of transparent objects. The G-Buffer contains information only about the nearest pixels, but we require multiple points per pixel when rendering transparent objects to correctly compute the final pixel color.

Due to the alpha blending equation:

$$color_{final} = (1 - \alpha_{src})color_{src} + \alpha_{src}color_{dst}, \quad (1)$$

which is used for correct rendering of transparent objects, the resulting color needs to be evaluated back-to-front from the camera's point of view. Therefore, a typical implementation sorts all triangles in the scene and then renders these back-to-front. Problems are pairs of intersecting

triangles which introduce the need to split at least one of those triangles into two parts.

For static transparent objects, constructing a BSP tree as a pre-processing step for all transparent triangles in the scene is a common approach. During scene rendering, the BSP tree is traversed back-to-front based on camera location. However, when we consider dynamic objects, the BSP tree needs to be maintained every frame (in worst-case, the whole tree needs to be rebuilt), which is usually a CPU intensive approach, especially for thousands of triangles.

4.1 Stencil Routed A-Buffer

Our goal was to have fully dynamic scenes, therefore we chose Stencil Routed A-Buffer [10] for rendering transparent objects. Thanks to newer hardware supporting OpenGL 3 and higher, it is possible to render to a multisample framebuffer object (FBO) and use the stencil buffer to fill each sample of a pixel with different values at different depths. This feature is called *Stencil Routing*.

At first, a multisample framebuffer with a depth, stencil and color buffer is created - this will be our Alpha buffer (A-Buffer). Then, every frame, the stencil values are initialized to $n+1$ for the n-th sample of a pixel by rendering a full-screen quad once for every sample while allowing writing only into the current sample.

During the rendering of transparent objects (with depth write disabled), we set the stencil operation to decrease whenever a fragment is being rendered and we set the stencil function to equal with the reference value 2. When the first fragment of a pixel is being rendered, the first sample (which has a stencil value of 2) is filled with color and depth of the fragment and all stencil values are decreased by one. Now the second sample has a stencil value of 2 and the next fragment being rendered for this pixel gets stored in the second sample. This behavior is shown in Figure 5.

Using this approach, we can compute n layers of transparent objects in one rendering pass, where n is the number of samples per pixel in our multisample FBO. Latest hardware allows up to 32 samples per pixel, but older graphic cards support 8 samples with no problems.

Finally, we need to display the transparent objects on the screen. We render a fullscreen quad and in the fragment shader, we access all the samples one by one and sort them based on their depth value. Finally, the sorted samples are blended in the fragment shader using standard alpha blending and the result is displayed on the screen. A result is shown in Figure 4.

Despite the improvement on speed of rendering, there is still a problem with shading transparent objects. When blending the fragments together, we need to shade them accordingly, and this is a forward shading step, so it inherits all the limitations of forward shading. Therefore, only a small amount of lights can be selected for shading the transparent surfaces.

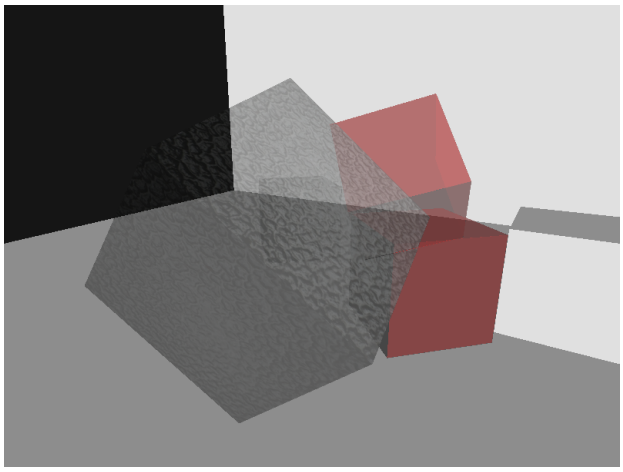


Figure 4: Stencil Routed A-Buffer - Result

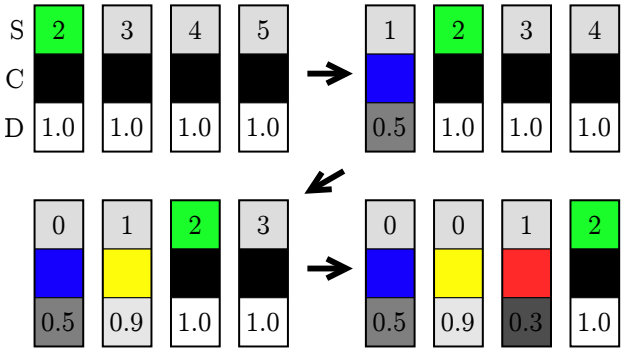


Figure 5: Stencil Routing - The 4 samples for a pixel after initialization step ready to write to the sample with stencil value $S = 2$ (top-left). When the first fragment is stored all stencil values are decreased by one, color C and depth D of the first fragment are written into the first sample (top-right) and after next two steps (bottom-left, bottom-right) we have three valid samples that can be blended together.

Problems can still occur when there are more samples per pixel than the multisample buffer can hold. Samples get lost and there is the question of what to do when an overflow occurs. One simple improvement is to use bounding volumes which are probably used during view frustum culling to estimate distance from camera and render sorted (not completely like with a BSP tree) geometry, which causes the closest surfaces to be stored in our A-Buffer before it overflows. Afterwards, overflown pixels will probably be more distant and their final contribution might be so small that the artefacts will not be visible.

5 Ambient Occlusion

Ambient occlusion is defined as the amount of ambient light reaching a point in the scene. It is calculated by integrating the visibility function over a hemisphere centered at the target point and oriented according to the surface

normal. Static ambient occlusion can be precomputed for static objects (and dynamic lights, since the occlusion does not depend on light sources), however it is a costly process and it does not produce correct results when there are moving objects in the scene.

5.1 Screen-Space Ambient Occlusion

Dynamic pre-computation of ambient occlusion is impossible especially when we don't know how objects in the scene will move. Therefore, we need a dynamic solution that estimates occlusion every frame.

In Mittring's work [9], the author proposed a method called Screen-Space Ambient Occlusion (SSAO) which uses the depth buffer (as rendered during the G-Buffer phase) to approximate scene geometry and estimate occlusion based on this inaccurate approximation.

The output of the algorithm is a one-channel 8-bit *accessibility texture* covering the entire screen. The value in each pixel in range $[0,1]$ is then used as a multiplication factor for ambient light of the respective pixel.

The SSAO generation pass occurs after the G-Buffer is generated and before we synthesize lighting, since we want to use the occlusion values during the lighting phase. We render a fullscreen quadrilateral and access the G-Buffer depth and the G-Buffer normal. For every pixel p with (x,y) coordinates and d depth, its 3D eye-space position P is reconstructed using the depth value and position in the depth map. In a small sphere around this 3D point, a number of points $Q_0,...,Q_c$ are generated by adding random vectors of length less than 1 multiplied by the sphere's radius r to the point's position.

Afterwards, every point Q_i is projected back into clip space which gives us (x_i,y_i) coordinates to the depth map and the actual depth d_i of point Q_i. If the value stored at position (x_i,y_i) in the depth map is smaller than d_i, there is an object covering point Q_i and it is a potential occluder for point P.

Our approach utilizes the G-Buffer normal as well, which increases the number of correct samples - samples that are below the surface are false occluders. This is not included in the Crytek implementation and it avoids self-occlusion which generates occlusion values of 0.5 on flat surfaces. In Figure 6, we have the scene's depth buffer as seen from the camera C. Point Q does not lie in the half-space assigned by point P and the normal at P. Therefore, any object containing Q should not be considered as an occluder. The object in front of Q should be considered and we rely on the point distribution that at least one of the sampling points will be inside the sphere and the sphere will contribute to occlusion. Even if this is not the case for one point, for neighbouring pixels it probably will be and after blurring the occlusion value gets propagated from those as well.

To avoid generating points behind the surface, we test the angle between the surface normal and the offset vector

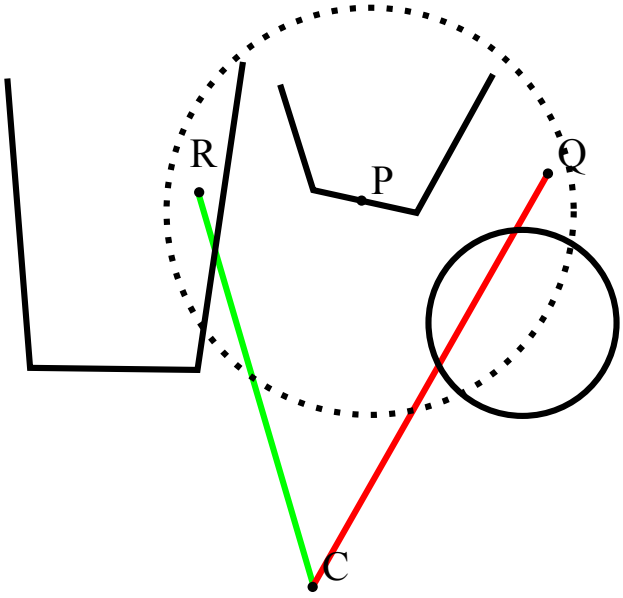


Figure 6: The SSAO algorithm - When calculating occlusion for the surface point P, we generate points such as Q and R in the dotted sphere with radius r. Both Q and R have depth larger than the value stored in the depth buffer, therefore both points are considered as occluders.

$\vec{v}$ added to P. If larger than $\frac{\pi}{2}$ we simply take $-\vec{v}$ as the offset vector.

For every potential occluder, the occlusion factor is computed as a function of the occluder's distance d:

$$O(d) = \frac{1}{1+d^2}. \tag{2}$$

A necessary step during the SSAO computation is the generation of random vectors that are added to P. If for every pixel we use the same set of vectors, certain repeating patterns occur. A more acceptable solution is that the vectors differ for neighboring pixel. We use a RGB texture containing random values that are interpreted as a vector and up to 32 uniformly distributed vector (stored as uniform variables during shader execution) in a unit sphere. In the shader, all the uniformly distributed vectors are reflected by using the vector value read from our random vector texture. This ensures different rotation of vectors on the sphere for neighboring pixels.

The difference can be seen in Figure 7. The randomized pattern gets blurred into a smoother result since it avoids large portions of uniformly occluded areas, which would not disappear after blurring.

The SSAO fragment shader performs a lot of operations. However, the results are quite noisy even when using a large number of samples and they still require a blurring step. Therefore, using a $\frac{w}{2} \times \frac{h}{2}$ (where $w \times h$ is the current resolution) sized SSAO texture is acceptable because it will be blurred anyway. Some implementations even use $\frac{w}{4} \times \frac{h}{4}$ SSAO textures.

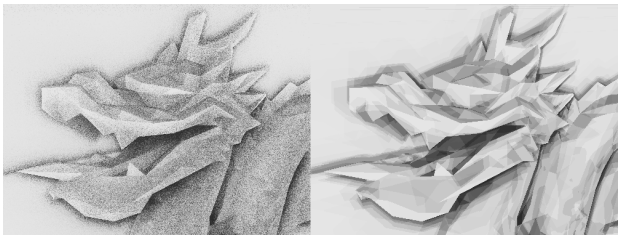


Figure 7: The difference when using randomized vector in SSAO (Left) and same for each pixel (Right)

5.2 Cross-Bilateral Filtering

Actual occlusion is mostly smooth and without any kind of noise. Therefore, we need to reduce noise somehow. Simply blurring the SSAO texture with a gaussian blur is not enough, because occlusion will "leak" through edges in the screen which results in an unnatural behavior such as the detaching of the occlusion in Figure 8.

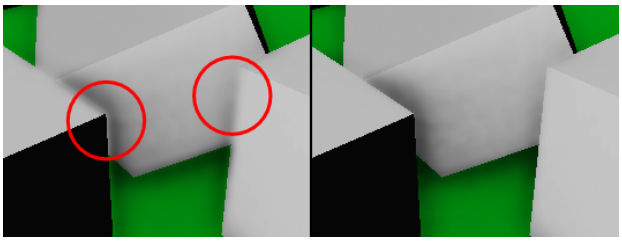


Figure 8: Artifacts occuring when using gaussian blur (Left) and elimination of those artifacts when using Cross-Bilateral Filtering (Right)

To avoid blurring over edges, we use a modified gaussian blur. What we want is an edge-aware blur that does not blur edges in the image. A great help is the depth buffer we have stored. The difference of depth values between two neighbouring pixels describes quite well whether there is an edge between these two pixels or not. If there is a tiny difference, the neighbouring pixels were very close to each other before projection.

We use a version of cross-bilateral filtering [14] coupled with a separable gaussian blur. We perform a horizontal and vertical blurring pass on the SSAO texture while multiplying the gaussian coefficients with a simple function of depth difference between the center sample d_{center} and the current sample $d_{current}$. We use the following function:

$$w(current) = \frac{1}{\delta + |d_{center} - d_{current}|}, \qquad (3)$$

$\delta > 0$ is a small constant to avoid division by zero.

After multiplying all samples with their corresponding weights, the sum of all weights is computed so we can normalize the result.

The results of SSAO after performing a simple gaussian blur and a cross-bilateral blur are shown in Figure 9.

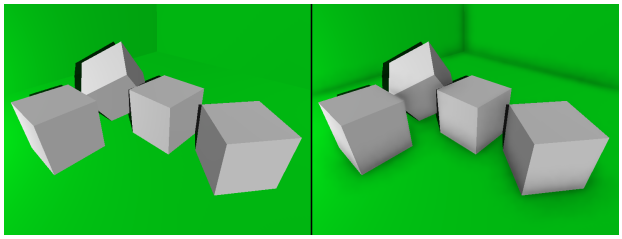


Figure 10: Comparison of a final scene without SSAO (Left) and with SSAO (Right)

In Figure 10, you can see the visual improvement provided by SSAO with a 15x15 cross-bilateral filter.

6 Complete rendering pipeline

Our implementation was written in C++ and OpenGL and runs on hardware supporting OpenGL 3.3. When putting together all the rendering parts described in this paper, the steps of our pipeline are as follows:

1. Render non-transparent objects while generating the G-Buffer.
2. Using the G-Buffer depth and normal textures, generate the SSAO accessibility texture.
3. Render transparent objects using Stencil Routed A-Buffer into a multisample buffer and perform forward shading on all pixels with the selected lights.
4. Render light volumes while generating the L-Buffer. Access SSAO texture for ambient term.
5. Blend sorted samples from the multisample buffer into the L-Buffer.
6. Compute log-average luminance and L_{white} from the L-Buffer.
7. Prepare unfiltered bloom texture by subtracting values based on average luminance.
8. Compose final bloom image by averaging multiple mipmaps.
9. Tonemap L-Buffer and display on screen.
10. Display Bloom on top of tonemapped data.

7 Performance and Results

We tested our rendering engine on several scenes, the testing machine had an Intel Core i5 750 CPU, a NVIDIA GeForce 9800 GT graphic card and 4GB of RAM. Our application is single-threaded at the moment, it was utilizing one core of the CPU.

We used three different quality settings of our engine during the tests:

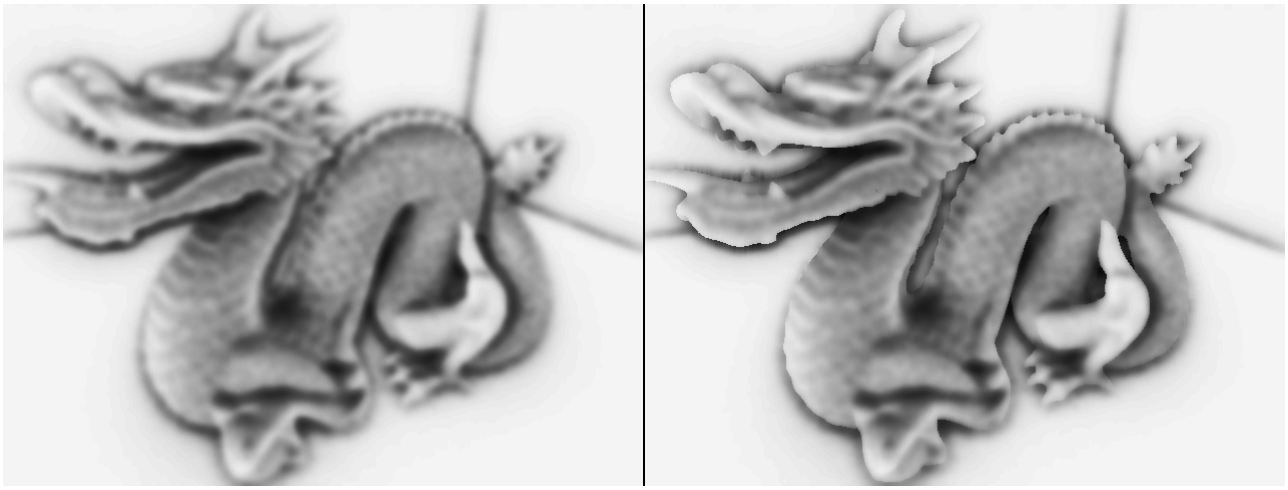


Figure 9: Comparison of simple blurring (Left) and edge-preserving blurring (Right) of the accessibility texture. Both images have been darkened for the effects to be visible. Most notable differences are around the dragon's head

- **Low** - 800x600 resolution, using $\frac{1}{16}$ sized SSAO texture with 16 samples per pixel and a 21x21 pixel wide bilateral blur. Stencil Routed A-Buffer had 4 samples per pixel. Using one 512x512 shadow map for sunlight.
- **Medium** - 1280x720 resolution, using $\frac{1}{4}$ sized SSAO texture with 16 samples per pixel and a 61x61 pixel wide separable bilateral blur. Stencil Routed A-Buffer had 8 samples per pixel. Using one 1024x1024 shadow map for sunlight.
- **High** - 1920x1080 resolution, using full-sized SSAO texture with 16 samples per pixel and a 61x61 pixel wide separable bilateral blur. Stencil Routed A-Buffer had 8 samples per pixel. Using one 2048x2048 shadow map for sunlight.

Furthermore, we used multiple presets to determine which operations take how much performance.

- **SSAO + SR** - SSAO and transparent objects enabled
- **SSAO** - SSAO enabled and transparent objects disabled
- **SR** - SSAO disabled and transparent objects enabled
- **OFF** - SSAO and transparent objects disabled

The results are shown in Table 1, listing average FPS values. SSAO is the main time-consuming operation, especially on High settings when using a full-sized buffer. However, on lower settings, the effects of SSAO were still visually appealing and only an experienced user would notice some artifacts due to the low resolution of the buffer. We do not recommend doing SSAO in full-size, $\frac{1}{4}$ sized buffer is a good performance and quality trade-off.

Note that the performance of SSAO depends mainly on screen resolution and samples per pixel. Since it is a screen-space algorithm, it does not in any way depend on scene complexity. This is one of the advantages of SSAO, that it provides stable performance and no unexpected FPS spikes.

Stencil Routed A-Buffer, on the other hand, depends on how many transparent objects and layers are visible at the moment. The more pixels, the more we need to sort and the higher FPS spikes. In the Dragon scene, there were no transparent objects, therefore the measurements are omitted.

All testing scenes had 6 animated point lights and one static directional light with Shadow Mapping as a good base for Deferred Shading.

8 Conclusion

We have presented a fully-dynamic real-time rendering system that overcomes any kind of pre-processing steps and allows dynamic objects and lights. The system runs at interactive frame-rates on newer hardware but it is compatible with OpenGL 3 hardware and it can be altered (in terms of quality) to run at interactive frame rates on older hardware as well. We have presented how the different techniques fit together and provide visually appealing quality.

Our system still has limitations to overcome, especially allowing an arbitrary number of lights to affect transparent objects without a performance hit. Other Real-time Ambient Occlusion techniques as well as Per-pixel Linked Lists for Order Independent Transparency should also be integrated into the system to evaluate the quality/speed trade-off and provide other solutions for latest hardware.

Scene	Triangles	Preset	Low	Medium	High
Sponza	287 353	SSAO + SR	18.7	10.9	3.0
		SSAO	20.9	13.8	3.6
		SR	19.7	15.8	10.7
		OFF	24.7	21.3	14.6
Dragon	201 075	SSAO	38.1	19.1	3.8
		OFF	49.6	36.2	21.2
House	11 563	SSAO + SR	46.7	18.3	3.9
		SSAO	60.1	26.0	4.6
		SR	67.5	32.7	17.8
		OFF	95.0	60.0	36.3

Table 1: Overall performance on three different scenes. Showing average FPS values for different settings of our engine

9 Acknowledgements

We would like to thank Marko Dabrovic for providing the Sponza Atrium model and The Stanford 3D Scanning Repository for providing the Dragon model. We also thank Martin Madaras and Andrej Ferko for providing the hardware for testing.

References

[1] L. Bavoil and K. Myers. Order independent transparency with dual depth peeling. Technical report, NVIDIA Developer SDK 10, February 2008.

[2] Michael Bunnell. Dynamic ambient occlusion and indirect lighting. In Matt Pharr, editor, *GPUGems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 223–233. Addison-Wesley, 2005.

[3] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a VLSI system for high performance graphics. *SIGGRAPH Comput. Graph.*, 22:21–30, June 1988.

[4] Cass Everitt. Interactive order-independent transparency. Technical report, NVIDIA Corporation, May 2001. Available at http://www.nvidia.com/.

[5] Dominic Filion and Rob McNaughton. Effects & techniques. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 133–164, New York, NY, USA, 2008. ACM.

[6] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pages 41–48. ACM Press, 2005.

[7] Abraham Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl.*, 9:43–55, July 1989.

[8] M. McGuire. Ambient occlusion volumes. In *Proceedings of the Conference on High Performance Graphics*, HPG '10, pages 47–56, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[9] Martin Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[10] Kevin Myers and Louis Bavoil. Stencil routed a-buffer. In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[11] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 267–276, New York, NY, USA, 2002. ACM.

[12] O. Shishkovtsov. Deferred shading in S.T.A.L.K.E.R. In Matt Pharr and Fernando Radima, editors, *GPU Gems 2*, chapter 9, pages 143–166. Addison-Wesley Professional, 2005.

[13] Nicolas Thibieroz and Holger Gruen. OIT and indirect illumination using DX11 linked lists. In *GDC San Francisco 2010*.

[14] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society.

[15] Michal Valient. Deferred rendering in killzone 2. Online, accessed Feb. 20th, 2012, 2007. Develop Conference, http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf.

[16] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12:270–274, August 1978.

Fast Random Sampling of Triangular Meshes for Hair Modeling

Martin Šik*
Supervised by: Jaroslav Křivánek†

Faculty of Mathematics and Physics
Charles University in Prague

Abstract

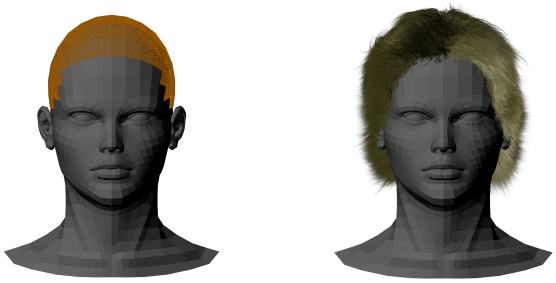
Efficient modeling of hair for realistic computer animation is a difficult problem because of the sheer number of individual hairs on a human head or an animal body. Random placement of hair roots on an arbitrary triangle mesh is an important sub-task of this problem. The main contribution of this paper is a simple, fast, and memory-efficient algorithm for randomly distributing points on a triangular mesh with a density specified by a two-dimensional texture. Our algorithm is many times faster than existing alternatives, such as rejection sampling. Furthermore, we describe a software architecture for procedural generation of hair in render-time. This module can generate millions of hairs during rendering from only a few guide hairs directly modeled by a 3d artist, which makes the rendering process very efficient.

Keywords: Mesh sampling, sample density, hair modeling, procedural generation

1 Introduction

Recent 3D animated films contain creatures with lots of fur and nowadays the trend in the film industry is to use CGI for such creatures even in non-animated films. The key question is how to effectively create hair (or fur) styling. Modifying each hair individually by a 3d artist is not practical because it would take too much time and therefore cost large amount of money. It is much easier for a 3d artist to create and model only a subset of hair fibers from which final hair will be automatically generated. The current approach is to export the generated hair fibers of the final hair into a large file that is then sent to the renderer that produces the high quality frames. However, because of the sheer number of individual hair fibers in the final hair style, such a file can take up gigabytes of disk space and therefore working with it becomes inefficient.

In this paper, we present a library that is able to generate final hair from a modeled subset of hair during render time, therefore skipping the export of hair to a scene file. Furthermore our library is able to display thousands of hair interactively. Hair generated by our library has also a number of parameters that improve realism of hair, such as creation of hair strands or influencing hair shape with a noise.



(a) Point samples (b) Generated hair

Figure 1: The left picture shows 100,000 point samples that has been used in the right picture as hair roots positions.

One of our goals is to have very fast generation of hair. The main bottleneck in hair generation is random placement of hair positions on a model. To remove this problem, we have developed a new algorithm for fast sampling of triangular meshes. Since real hair density may vary a lot over a human or an animal skin, the main feature of our sampling algorithm is that the density of generated samples can be defined by a two-dimensional texture mapped on the model from which the hair grows.

Figure 1 shows an example of hair roots placement. Image 1a demonstrates 100,000 points generated by our algorithm while the image 1b shows hair procedurally generated by our library growing from these points.

In Section 2, we discuss the present state of the art in mesh sampling algorithms and hair modeling. In Section 3, we describe our new algorithm for sampling on triangular meshes and present its results. In Section 4, we present a software architecture for procedural generation of hair. Finally, we conclude in Section 4 and present directions for future work.

2 Related Work

2.1 Random Sample Generation

Random sample generation is a problem that has been a point of interest in the field of computer graphics for a long time, since it can benefit a variety of graphics applica-

*martin_sik@centrum.cz
†jaroslav@cgg.mff.cuni.cz

tions in texturing, rendering, remeshing, and point-based graphics [6, 7, 10]. Our goal is fast generation of samples on a triangular mesh. This specific issue is addressed by [2], however their priority is not efficiency, but rather good distribution of generated samples. Apart from this article there are other works (for example [4]) that address fast generation of samples with good distribution, but they generate samples in a plane and it is unclear how to apply them for generation of samples on a triangular mesh. Standard methods like rejection sampling [6] can also be used to generate samples on a triangular mesh, however they are too slow for our purpose.

2.2 Modeling and Rendering Hair

Modeling and rendering of hair is an issue addressed by numerous works, but they do not discuss the procedural generation of hair in render-time. Among the tools used for hair modeling in standard modeling programs (such as *Autodesk Maya*), *Shave and a Haircut*[1] allows the user to model a subset of hair from which it then procedurally generates hair. However *Shave and a Haircut* is unable to execute the generation of hair in render-time without the need of saving hair geometry to a scene file. Since *Shave and a Haircut* is a commercial product, it is unknown how it generates hair or places hair roots on a model. Generation of hair in render-time is mentioned in [8], however their work is focused on speeding up rendering of hair in a specific renderer and they omit how exactly is the generation of hair done.

3 Random Points Generation

In this section we describe our algorithm for randomly distributing points on a triangular mesh with a density specified by a two-dimensional texture. Note that the problem would be trivial if the desired point density was uniform over the surface: in such a case, we would simply pick a triangle proportionally to its area (using the inversion of a discrete cumulative distribution function) and place the point uniformly in the selected triangle.

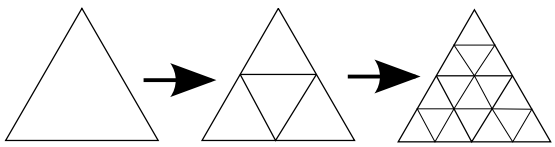


Figure 2: Recursive subdivision of a triangle.

However, since the point distribution probability density must take into account both the density texture mapped on the mesh surface and also each mesh triangle's area size, creation of the cumulative distribution function is not trivial. We overcome this problem by recursively subdividing each mesh triangle to sub-triangles (see Figure 2) until every sub-triangle's surface has uniform density. We then calculate the probability that new sample will be created on a given sub-triangle as the density texture value integrated over the sub-triangle's surface. We use these probabilities to create the discrete cumulative distribution function. Finally for every desired random point we first use this cumulative distribution function to randomly select any sub-triangle from the mesh surface based on its probability and then we uniformly sample the selected sub-triangle to determine a generated point position.

[1] http://www.joealter.com/

3.1 Defining the sub-triangle probability distribution

In order to define the sub-triangle probability distribution we iterate through all triangles of the given mesh surface. As previously mentioned, we recursively subdivide each triangle, however to make the subdivision a lot faster we do not actually check if every sub-triangle's surface has uniform density, instead we will stop the subdivision if no more than one texel of the density texture is mapped on each sub-triangle. Thanks to this we can calculate a subdivision depth for each triangle directly from number of the density texture texels mapped on it. Also we do not need to integrate the density texture over a sub-triangle to determine its probability, we only evaluate the density texture at the sub-triangle's barycenter.

As we will discuss later, both the speed and memory cost of our algorithm depends on the total number of sub-triangles. To decrease the number of sub-triangles we can check the probabilities of four sub-triangle siblings and if they have the same probability, we can use their parent instead of them (see Figure 3). Again we can check the parent's siblings and continue until the probabilities differ or we have reached a non-subdivided triangle. After these steps are applied each sub-triangle is only divided if its surface has not uniform density as it was mentioned in the brief algorithm description.

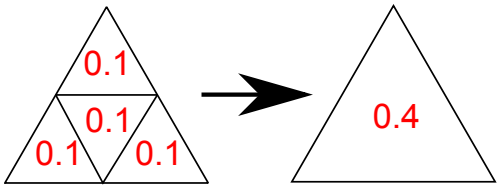


Figure 3: If the four sub-triangle siblings have the same probability, their parent may be used instead of them. The numbers represent the probabilities.

As we have said before, the final step of our algorithm is sampling the selected sub-triangle. In order to calculate the sample point position on the triangle mesh we need to store for each sub-triangle a reference to the parent mesh triangle and where it is located inside the triangle (i.e. the barycentric coordinates of the sub-triangle in the parent triangle). This has to be done during the computation of the

sub-triangle probability distribution. Since the subdivision scheme is same for every triangle, we only need to give unique index to each possible sub-triangle position inside a triangle (see Figure 4) and store barycentric coordinates of these sub-triangle positions in a separate data structure, where they can be easily accessed by the position index. This significantly reduces memory consumption, since we only need to store two indices per sub-triangle (a parent triangle index and a sub-triangle position index) and its probability and therefore each sub-triangle takes up only 12 bytes in memory (considering the fact that all 3 values takes up 4 bytes). The size of the pre-computed data structure which stores sub-triangle positions is negligible.

During a sub-triangle subdivision we will need to calculate the index i of a sub-triangle position from the index i_{parent} of its parent sub-triangle position: $i = 4(i_{parent} + 1) + j$, where $j \in [0,3]$ is the j-th sub-triangle of its subdivided parent sub-triangle.

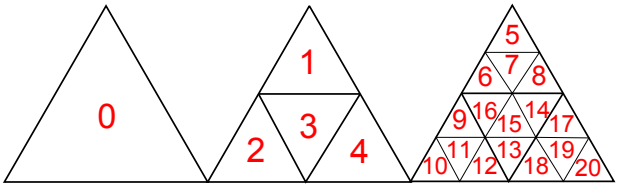


Figure 4: Indices of sub-triangle positions inside a triangle.

The complete algorithm for the computation of the sub-triangle probability distribution is described in Figure 5.

3.2 Generating a Point Sample

As mentioned before, to generate a point sample we first need to randomly select a sub-triangle based on its previously computed probability. To implement this random selection we need to convert sub-triangles probabilities to a cumulative distribution function.

The cumulative distribution function is defined as $F = P(X \leq x)$, where $P(X \leq x)$ is the probability of $X \leq x$ and X is a random variable from the distribution D. Since we use only discrete version of the cumulative distribution function, we can easily compute it:

$$F(j) = \sum_{i=0}^{j} P(X = i) = F(j-1) + P(i),$$

where $F(0)$ is defined as $P(X = 0)$ and $P(X = i)$ describes the probability of a discrete random variable X being equal to i. The discrete function F is then stored as an array.

Following [6], we find a sub-triangle as $\arg\min_i (F(i) > \xi)$, where ξ is a random number from $U(0, \max F(x))$. Because values of $F(i)$ are increasing with greater i, we can use the bisection algorithm to find $\arg\min_i (F(i) > \xi)$. Finally, we generate a random sample in the selected sub-triangle with uniform probability density, and map the sample to the barycentric

For each mesh triangle T:

1. Calculate the area of T. The subdivision depth i is now 0. Mark the triangle T as the sub-triangle D_0 (D_i denotes sub-triangle in the subdivision depth i).
2. **If** the subdivision depth i is less than maximum (more than one texel of the density texture is mapped on the sub-triangle D_i):
   (a) Subdivide the sub-triangle D_i to four smaller sub-triangles D_{i+1}.
   (b) For each sub-triangle D_{i+1} store the index of the triangle T and the sub-triangle D_{i+1} position inside T.
   (c) Increase the depth of recursion i by one and call step **2.** for every sub-triangle D_{i+1} of the sub-triangle D_i.
3. **Otherwise** (the maximum subdivision depth was reached):
   (a) Calculate the probability P_{D_i} of the sub-triangle D_i as the area of D_i multiplied by the density texture value mapped on D_i's barycenter.
   (b) **While** each of four sub-triangles D_i with the same parent sub-triangle D_{i-1} are not subdivided and have the same probabilities P_{D_i}:
      i. Discard sub-triangles D_i and use their parent D_{i-1} instead with the probability $P_{D_{i-1}} = 4P_{D_i}$.
      ii. Decrease the subdivision depth i by one and if $i = 0$ exit the while-cycle.
   (c) Store sub-triangles probabilities P_{D_i}.

Figure 5: The computation of the sub-triangle probability distribution.

coordinates of the parent mesh triangle. The complete algorithm for generating a point sample is described in Figure 6.

3.3 Further Improvements

As described in Figure 6, for each sample we select a sub-triangle using the bisection algorithm. The bisection algorithm runs in logarithmic time proportional to the size of the cumulative distribution function domain. Since the domain of F is usually very large, even logarithmic time for generating each sample may be quite a lot.

We improve speed of the sub-triangle selection by creating 1-dimensional uniform grid G over the F codomain (see Figure 7). For each cell C of the grid we store two indices C_{begin} and C_{end} (elements of the F domain):

$$\begin{aligned} C_{begin} &= \arg\min_i \{F(i) \in C\} \\ C_{end} &= \arg\max_i \{F(i) \in C\} \end{aligned}$$

When generating a sample, we first generate a random number $\xi_0 \in [0, \max F]$ as before, but then we determine a cell C of the uniform grid for which $\xi_0 \in C$ in constant time, after that we select a sub-triangle using the bisection algorithm only in limited domain of F: $[C_{begin}, C_{end}]$.

For every requested point sample:

1. Select a sub-triangle:
   (a) Generate a random number $\xi_0 \in [0, \max F]$.
   (b) Select a sub-triangle as $D = \arg\min_i F(i) > \xi_0$ using the bisection algorithm.
2. Generate a sample in the selected sub-triangle
   (a) Generate two uniform numbers $\xi_1, \xi_2 \in$ [0,1].
   (b) Calculate the barycentric coordinate $u_D = 1 - \sqrt{\xi_1}$
   (c) Calculate the barycentric coordinate $v_D = \xi_2 \cdot \sqrt{\xi_1}$
3. Map the sample to the barycentric coordinates (u,v) in the parent mesh triangle:
   (a) From u_D, v_D calculate the barycentric coordinates of a sample in the triangle T containing the selected sub-triangle D:
   $u = u_D \cdot u_1 + v_D \cdot u_2 + (1 - u_D - v_D) \cdot u_3$
   $v = u_D \cdot v_1 + v_D \cdot v_2 + (1 - u_D - v_D) \cdot v_3$
   where u_i, v_i are the barycentric coordinates of D vertices inside T. We obtain u_i, v_i by a lookup in the precomputed sub-triangle position data structure (Section 3.1) using the sub-triangle index.

Figure 6: Generating a point sample.

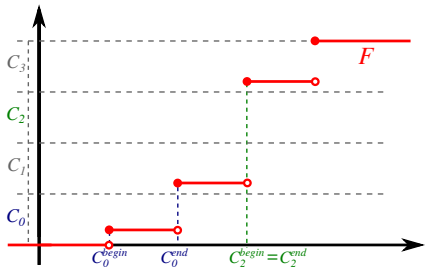


Figure 7: The uniform grid G built over the cumulative distribution function F codomain. The C_{begin} and C_{end} represent indices stored for every grid cell C.

The number of cells in the uniform grid G will influence samples generation performance. If grid has so many cells that in each cell lies only single member of the F domain then the selection of a sub-triangle will run in constant time at the cost of an increased memory consumption.

3.4 Results

We test our sampling algorithm on a single core of a 3.07 GHz PC with 6 GB RAM running Windows 7 64bit. We typically use the same number of cells of the uniform grid as is the size of the F domain (i.e. the number of generated sub-triangles) and a density texture with resolution 1024×1024 except if noted otherwise.

3.4.1 Comparison to Rejection Sampling

We compare our algorithm against rejection sampling since we were not able to find any other alternatives in the existing literature. The rejection sampling first randomly selects a triangle based on its area (for that purpose we use in our implementation a cumulative distribution function), then the triangle is uniformly sampled and the sample is accepted if a random value is not lesser than the value of the density texture at the sample point, otherwise it is rejected and rejection sampling generates a new sample.

Figure 8 plots the performance of our algorithm and the rejection sampling algorithm tested for three models with a uniform density texture. Since we have used a uniform density texture, the rejection sampling algorithm never rejects any generated sample. Faster sampling rate of our algorithm is therefore caused only by the usage of the uniform grid built over the cumulative distribution function codomain: we can see that this technique provides a speed-up between 3 and 6 for the tested cases. Usage of the uniform grid should remove the dependency on a model triangle count, however if triangles' areas differ greatly the cumulative distribution function is non-linear and therefore the uniform grid is not very efficient. All models are displayed in Figure 10 and their triangle counts can be found in table 2.

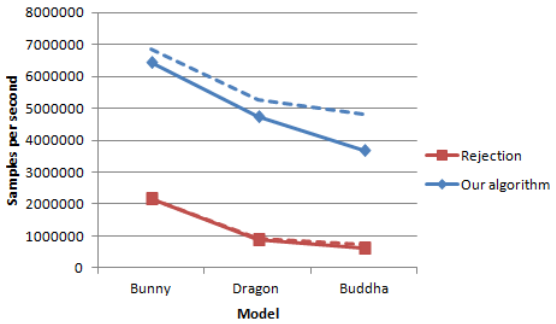


Figure 8: The sampling rate (samples per second) for our algorithm and for the rejection sampling algorithm. 3 different models with a uniform density texture were used in this test. Dashed lines represent the sampling rate without the preprocess time (i.e. triangle sub-division and the cumulative distribution function construction).

We have also tested the influence of the uniform grid resolution on the sampling performance. As shown in Table 1, total time spent on the preprocess part of our algorithm is only slightly influenced by the cell count, however memory consumption grows significantly with increasing number of the grid cells. The highest sampling rate is achieved when the number of cells is two times higher than the size of the cumulative distribution function domain.

Figure 9 also plots the performance of our algorithm and the rejection sampling algorithm, but this time we have tested them with 4 density textures with different average values. The average value of a density texture corresponds to the overall density of samples and also to the percentage

Grid size	#Samples	CDF constr.	Memory
0%	2365628	0.054s	1.33 MB
13%	5855405	0.055s	1.47 MB
25%	5951817	0.057s	1.60 MB
50%	5890408	0.058s	1.86 MB
100%	6387220	0.059s	2.39 MB
200%	6531778	0.061s	3.45 MB
400%	6447026	0.066s	5.57 MB

Table 1: The results of testing the preprocess part of our algorithm and sampling speed for the bunny model with different resolutions of the uniform grid. The grid size is reported as percentage of the cumulative distribution function size. We have used a uniform density texture for this test.

of samples accepted by the rejection sampling algorithm. The plots show that the performance of our algorithm remains roughly the same for all textures, but the performance of rejection sampling algorithm depends linearly on a decreasing texture average value.

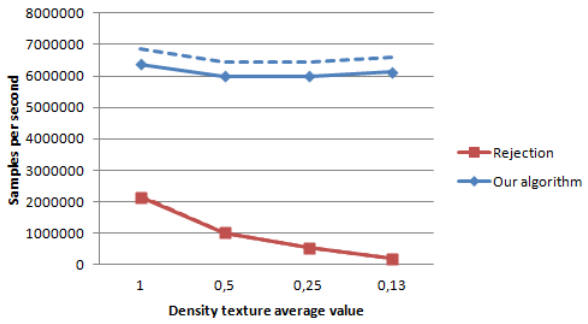


Figure 9: The sampling rate (samples per second) for our algorithm and for the rejection sampling algorithm. 4 density textures with different average values mapped on the bunny model were used in this test. Dashed lines represent the sampling rate without the preprocess time.

3.4.2 Performance

Table 2 shows the preprocess time for three different models with a uniform density texture. Both time and memory consumed by the preprocess is roughly linear in the model triangle count.

Model	Preprocess	Memory	#△
Bunny	0.070s	2.39 MB	69k
Dragon	0.127s	8.68 MB	202k
Buddha	0.25s	51.26 MB	1087k

Table 2: The result of testing the preprocess part of our algorithm for mesh surfaces with different triangle count. The results of sampling performance are shown in Figure 9. All models were used with a Perlin noise density texture.

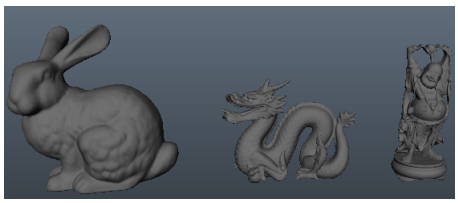


Figure 10: Models used for testing.

Finally we have tested the influence of the density texture resolution. Table 3 demonstrates that time and memory spent on the preprocess part of our algorithm depends linearly on a density texture texel count. Thanks to the uniform grid the sampling rate is only diminished by the longer preprocess time.

Resolution	#Samples	Preprocess.	Memory
512^2	6757501	0.019s	4.49 MB
1024^2	6306838	0.070s	5.33 MB
2048^2	4998126	0.253s	31.26 MB

Table 3: The results of testing our sampling algorithm for the bunny model with different resolutions of a Perlin noise density texture. The number of generated samples is per second.

3.4.3 Visual Results

Figure 11 shows a uniform distribution of points on the bunny model. The points distribution over the surface is not particularly good, but that is the price we have to pay for very fast sampling. Since the main purpose of this algorithm is the placement of hair roots positions, the quality of sampling is not so important. Figure 12 demonstrates a distribution of points controlled by a simple density texture.

Figure 11: The bunny model with 8,000 uniform points samples.

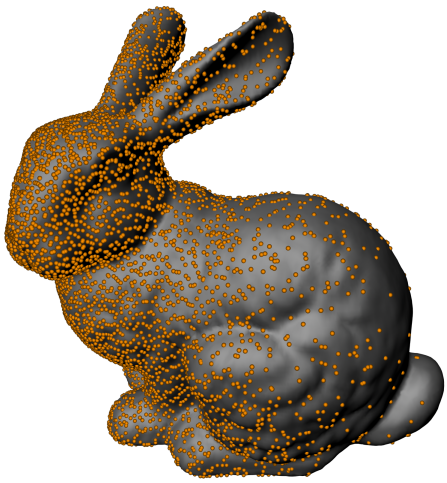


Figure 12: The bunny model with 8,000 points samples of increasing density from left to right.

4 Hair Modeling and Rendering

The procedural generator of hair we provide is a part of a complete hair modeling plug-in for Autodesk Maya called Stubble. We will first describe the pipeline of hair modeling used in Stubble in Section 4.1 and then we will closely discuss the software architecture of the hair procedural generator in Section 4.2.

4.1 Hair Modeling Pipeline

The main idea behind hair modeling with Stubble is to let a 3d artist model only small subset of hair, called hair guides, from which the rest of hair will be automatically generated by Stubble. Creating hair with Stubble can be divided into several steps:

1. **Preparing hair guides**: The first step is to create a few hair guides on a selected triangular mesh. The user may set the number of hair guides and their density over the mesh. Their roots positions are generated by our sampling algorithm described in Section 3.

2. **Modeling hair guides**: Each hair guide is represented by a polyline and user can model it using special tools which behave like comb or scissors.

3. **Applying dynamics**: After the basic modeling of guides is done, hair guides can be animated by applying hair dynamics.

4. **Hair properties**: When hair guides are properly modeled and animated, we procedurally generate the final hair fibers based on the guides. In the basic form, the generated hair simply interpolate their shape from nearby guides, but they may additionally be affected by random noise, color and other parameters. Stubble plug-in can interactively display thousands of generated hair during hair modeling in the Maya viewport, so the 3d artist may see how the final hair will look like. This is possible thanks to the efficiency of our point distribution algorithm.

5. **Rendering**: Final step is rendering hair with specialized rendering software. During rendering hundred thousands or even millions of hairs with selected properties are generated by Stubble from hair guides.

Figure 13 demonstrates hair modeling on a human head.

(a) Hair guides modeling (b) Interactive hair generation (c) Final render

Figure 13: Hair modeling pipeline

4.2 Procedural Generation of Hair

4.2.1 The Library Architecture

We have designed our procedural hair generator as a dynamic-link library that can be executed by different renderers. For each supported renderer, there is separate entry-point which implements renderer specific requirements for libraries that generate scene geometry.

To improve performance of hair generation we want to parallelize it. Since renderers are usually parallelized themselves, we take advantage of that and parallelize hair generation by calling our library separately from different threads of a renderer. Each of these calls is responsible for generating hair on one selected part of a triangular mesh and runs in a single thread. User of Stubble sets the number of parts to which the mesh is split and therefore the number of calls of our library. Is then up to the renderer to decide on how many threads these calls will be executed. It is important to mention that each frame of rendered animation is handled completely separately. For example if the mesh on which hair grows is split to 8 parts and we render 4 frames, our library will be called 32 times.

Most of the renderers require to know the bounding box of the geometry generated by any library before the library is even called. For example RenderMan interface compliant renderers use this information to optimize memory consumption, which is key feature when rendering millions of hairs. It is mentioned in [8] that it is for the best to calculate the tightest possible bounding box from complete hair geometry no matter additional time consumption. Therefore we generate hair geometry twice, first before rendering to calculate the bounding box and then during rendering we generate the hair again for actual rendering purpose. Since we have split hair generation during

rendering to several library calls, we have to calculate the bounding box for each call separately, which is done in parallel.

Generating hair twice could be avoided if we generated hair before rendering and then supplied it to a renderer directly with the rest of a rendered scene. This approach seems logical but it has one huge pitfall. We would have to save all hair geometry to a scene file. Since there can be millions of hairs, the scene file size could grow to several gigabytes for a single frame. Because the hair generation is very efficient (which is partly thanks to the speed of our point generation algorithm described in Section 3), handling such files causes much worse performance than generating hair twice, especially when these files usually have to be moved from a 3d artist workstation to computers dedicated only to rendering.

There are many parameters that influence hair generation. The modeling part of Stubble is responsible for saving them to files which are then used by our library during rendering. There are two types of these files. First of them stores properties like hair color or width that are shared among all library calls. The second type of file stores data specific for one library call, such as the selected part of the mesh and the number of hair generated by this library call. All input files are compressed to save disk space. Since Stubble enables interactive display of hair during modeling, hair parameters may be send to our library directly from the modeling part of Stubble without the need of creating any files.

4.2.2 Hair Generation

In this section we describe how every single hair is generated by our library. Before we start describing hair generation, it is important to know how we represent each hair. The most flexible way is to handle each hair as the Catmull-Rom curve, an interpolation curve defined only by the vertices it passes through. Furthermore, we specify for each of these vertices color, opacity, curve width and curve normal (an unit vector perpendicular to the curve tangent at a given vertex), which are then interpolated along the curve by a renderer.

The simplified flowchart in Figure 14 shows the generation of a single hair. It starts with creating a sample on the triangular mesh with a density defined by a texture as described in Section 3. The sample serves as the position of a hair's root. The generation of the sample must be very fast, otherwise it would be a bottleneck in the generation of hair.

When we know the position of hair, we generate its basic geometry by interpolation from few closest hair guides. To determine the closest hair guides we use euclidean distances of hair root from hair guides roots. To speed up this process, we store hair guides roots in a KD-Tree [3]. We have already mentioned that each hair is a curve defined by vertices and each hair guide is a polyline and therefore also represented by vertices. The hair curve and each hair guide

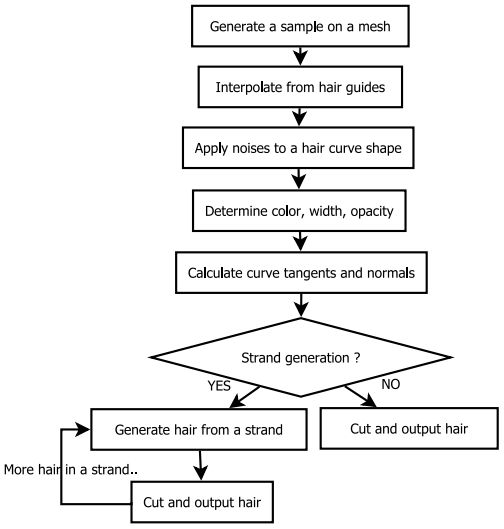


Figure 14: Flowchart of the generation of a single hair.

has the same number of vertices, so we can easily interpolate the hair curve by interpolating its vertices from corresponding hair guides vertices. To interpolate vertices we use *Scattered Data Interpolation*, specifically the Shepard method [9], which gives each guide hair a weight based on the distance from the interpolated hair.

Interpolating the hair curve from hair guides is not enough to make generated hair realistic, because the number of generated hair is far greater than the number of hair guides. To improve hair realism, we randomize each hair curve by adding random vectors generated from Perlin noise [5] to hair curve's vertices. The number of vectors used on a single hair, vectors size and noise frequency are user parameters. Figure 15 shows an example of a noise influence on hair.

Figure 15: An example of a noise influence on straight hair.

The next step is to define hair color, opacity and width. The user specifies these attributes at hair root and tip and we interpolate them to every hair vertex. Again we add some noise to hair color to increase realism.

Finally we have to calculate hair curve normals. Since curves are usually generated by a renderer as flat ribbons, we have to define the rotation of the ribbon about the curve. This is done by the curve normals. We use the method described in [11] to calculate reasonable normals. Computation of normals requires as input curve tangents, which are easily calculated for a Catmull-Rom curve from its vertices [1].

A single hair is now completely specified, we can either output it as is or we can use it to generate a whole

strand of hair. The user can specify if strands should be generated and how many hairs are in a single strand. If we generate a hair strand, we use the just generated hair as a basic hair around which all hair from this strand is generated. Each hair from strand inherits its properties from the basic hair and it is also influenced by the basic hair geometry. Furthermore, user can define several parameters of hair strands, that control the spread of hair roots and tips from the basic hair, twisting of hair around the basic hair and much more. Figure 16 demonstrates hair strands.

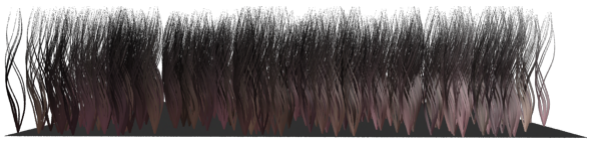


Figure 16: Generated hair strands.

Just before we output each hair to a renderer, we cut it. How much is each hair cut is defined by a texture mapped on the mesh from which the hair grows. The texture value specifies a curve parameter at which the curve is cut. Every user parameter controlling hair generation is also specified by a texture, which gives the user the ability to set different parameters for each hair.

Figure 17 shows hair generated by our library. There is 1,600,000 hair on this animal and it took only 3.2 seconds to generate it on a two core 3.07 GHz PC.

Figure 17: An animal with 1,600,000 hair generated by our library.

5 Conclusions and Future Work

In summary, we have presented a sampling algorithm suitable for fast sampling on triangular meshes. The density of samples generated by our algorithm is defined by a two-dimensional texture. In our tests, our algorithm achieves a 3 – 33 speedup compared to the fastest available alternative - the rejection sampling.

Furthermore, we have described a hair generator library that is able to generate millions of hairs in a few seconds during rendering. Our library utilizes the presented sampling algorithm for a very fast placement of hair roots. Hair generated by our library is influenced by several properties and by a few hairs modeled directly by a 3d artist. Our library is also able to display hair interactively in a modeling program.

There are several limitations of our work that should be addressed in future work. First, the uniformity of the generated samples should be improved. Second, we would like to add several modifications to our hair generator library, such as generation of low quality hair if an object with hair is motion blurred or is far away from a scene camera.

Acknowledgments

I would like to thank Jaroslav Křivánek for his useful comments and advice during the creation of this work.

References

[1] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., 2008.

[2] John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph.*, 29:166:1–166:10, 2010.

[3] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., 2001.

[4] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2006)*, 25(3):509–518, 2006.

[5] Ken Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, 2002.

[6] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., 2010.

[7] Lijun Qu and Gary W. Meyer. Perceptually driven interactive geometry remeshing. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 199–206, 2006.

[8] David Ryu. 500 Million and counting: Hair rendering on Ratatouille. Technical report, Pixar, May 2007.

[9] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, pages 517–524, 1968.

[10] Greg Turk. Texture synthesis on surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 347–354, 2001.

[11] Wenping Wang, Bert Jüttler, Dayue Zheng, and Yang Liu. Computation of rotation minimizing frames. *ACM Trans. Graph.*, 27:2:1–2:18, 2008.

Real-time particle simulation of fluids

Zsolt Horváth
Supervised by: Adam Herout

Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

Physically plausible simulation of fluids in real-time is mostly achieved using approximations of the Navier-Stokes equations. Recent methods simulate fluids by exploiting the capabilities of modern graphics processing units. This article describes a method called Smoothed Particle Hydrodynamics (SPH), which is a numerical approximation of the Navier-Stokes equations. The real-time simulation allows for interactivity which is a great advantage against the offline methods. Offline methods are not running in real-time. The main goal of this project was to experiment with the Smoothed Particle Hydrodynamics running in realtime on the GPU.

Keywords: particle systems, fluid simulations, Navier-Stokes equations, smoothed particle hydrodynamics, CUDA, GPGPU, marching cubes

1 Introduction

Real-time simulation of fluids is a hot topic and major challenge in computer graphics. Fluids like liquids and gases are an important part of our life and environment. In real-time grapics we tradicionally try to reproduce a part of our world as visually realistic as possible, but unfortunately it is hard to simulate. Researchers concentrate on developing new, better and faster methods to simulate and visualize fluids. It is commonly said to be one of the hardest phenomenon to simulate realistically and even harder to simulate detailed fluids interactively. The offline methods which run not in real-time, can generate physically and visually better results, but with no user interaction, which is a disadvantage.

The mostly used method in computer graphics to simulate fluids is the Smoothed Particles Hydrodynamics (SPH) [6]. SPH is based on the Navier–Stokes equations. Because of the complexity of these equations, they can be solved only in simple cases. Generally, they are solved by a numerial method. The SPH method has good approximation which computes the most important properties of fluids like density, pressure and viscosity.

The aim of this work was to experiment with the SPH method on the CUDA platform. The experiments were mostly focused on achieving a simulation running in real time with different types of visualization methods and on speeding up the slowest parts of the implemented algorithms.

This work is structured as follows. In Section 2 we describe the present state of methods used in this area. Then the Section 3 and 4 present theoretical aspects. Section 5 contains important implementation details. The results are summarized in Section 6.

2 Related work

In 1822 Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes equations that describe the motion of the fluid substances [3]. With these equations which describe the conservation of momentum together with the two additional equations for the mass and energy conservation, it is possible to simulate the fluid flow.

Simulations apply numerical methods to solve the nonlinear partial differential equations. One common way to do this is to treat the fluid as a continuum, discretize the spacial domain into a grid, and use the finite differences or the finite volume method [16]. In the literature about the fluid simulations, the grid-based fluid models are called Eulerian models. The fluid is thought of as being composed of fluid cells that form a uniform grid. Each cell contains a number of fluid molecules, or particles. The grid-based methods, as a matter of principle, have the drawback of a bounded simulation space which is caused by the finite memory of the computation devices. The fluid can not flow freely in the virtual environment because it can not exist outside the grid; it is locked in the grid.

The grid provides a solution to estimate the derivatives using a finite difference method (FDM). For theoretical details on Eulerian fluids see [5, 8, 19]. Although the Eulerian method provides better description of some the properties of fluids (mass-density, pressure field) compared to the Lagrangian method, but the major disadvantage is the grid itself.

The particle-based methods in the literature are called the Lagrangian models. These methods represent fluids using a discrete set of particles. These particles simulate the flow of the fluid by solving the particle hydrodynamics. For the real-time applications this has some advantages

against the grid-based methods. The biggest advantage is that the fluid can spread freely in the simulation space. For these reasons, we focus on the Lagrangian method based on the Smoothed Particle Hydrodynamics [6], which is the most popular solution for this kind of applications simulating the fluids. Each particle distributes its fluid properties to the surronding particles in the near neighbourhood using a radial kernel function (the smoothing kernel). The evaluated new property of a particle is the sum of property quantities of neighbour particles weighted by the smoothing kernel.

The Smoothed Particle Hydrodynamics method was introduced in 1977 by Monaghan and Gingold [15] and independently by Lucy [9]. First, it was used in astrophysics to simulate large scale gas dynamics [18]. Later, it was applied to incompressible flow problems. In the real-time applications, at first the Eulerian method was favoured. Müller, Charypar and Gross [12] showed that the SPH is well suitable for interactive real-time applications. Vizualization of the results plays as important role as the main simulation process in these applications. The most often used methods for rendering are these: point splatting [12], marching cubes or marching tetrahedra [21, 20], and raycasting [7].

3 Smoothed Particle Hydrodynamics

This section is focused on the theoretical explanation of the Lagrangian equations and how they are discretized. The most important parts are symmetrizations of the forces, like pressure and viscosity. The last part about the smoothing kernels describes the force computations.

3.1 Lagrangian equations

This subsection describes the Lagrangian method of the fluid simulation. The conservation of mass / continuity is given by:

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \mathbf{v}, \tag{1}$$

where ρ is density, $\mathbf{v}$ velocity and t is time. Using the substantive derivative [4], which specifies: $\frac{d\mathbf{v}}{dt} = \frac{\partial \mathbf{v}}{\partial t} + v \cdot \nabla v$, it defines the strength of how viscous the fluid is. We get the Lagrangian formulation of the conservation of momentum:

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P + \frac{1}{\rho}\nabla \cdot \tau + \mathbf{f}, \tag{2}$$

where v is the velocity field, ρ the density, P the pressure, $\mathbf{f}$ are external forces and τ the viscosity coefficient. We can ignore the mass conservation if we assume that the number of particles in constant. Finally end with this expression:

$$a_i = -\frac{1}{\rho_i}\nabla P_i + \frac{1}{\rho_i}\nabla \cdot \tau_i + \mathbf{f}_i = f_i^{pressure} + f_i^{stress} + f_i^{external}, \tag{3}$$

where a_i is the acceleration of a particle, $f_i^{pressure}$ is the pressure force, f_i^{stress} is the deviatoric stress (viscosity) and $f_i^{external}$ is the sum of external forces (e.g. gravity, boundaries). The remaining equations are derivable from previous equations or they can be found in [13, 11].

3.2 Discretization

The SPH can be used for any kind of fluid simulation. This is an interpolation method for the particle systems. In this method, the field quantities are only defined at discrete particle locations and can be evaluated anywhere in the space. For this purpose, the SPH distributes the property quantities in the neighbourhood of any particle using the smoothing kernel. In the SPH, a scalar quantity is interpolated at a specific location by a weighted sum of the contributions from all particles:

$$A_i(\mathbf{r_i}) = \sum_j m_j \frac{A_j}{\rho_j} W\left(\mathbf{r_i} - \mathbf{r_j}, h\right), \tag{4}$$

where j iterates over all particles, m_j is the mass of the particle, A_j is the scalar property, r_j is the position and h is the radius of the smoothing kernel.

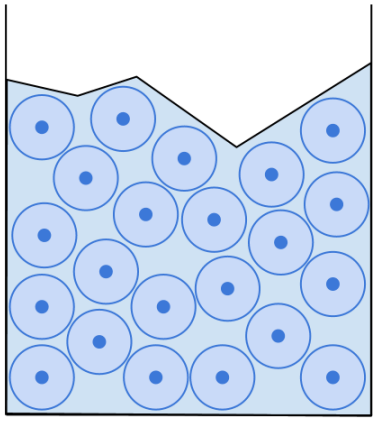


Figure 1: Lagrange particle-based fluid structure in 2D. The particles are represented by the dots. The circles represent the volume of each particle.

The function $W(\mathbf{r},h)$ is called the smoothing kernel with the core radius h. The W must be even $(W(\mathbf{r},h) = W(-\mathbf{r},h))$ and have finite support. If the W is even and normalized, the interpolation is of second order accuracy. The kernel is normalized if the following is true:

$$\int W(\mathbf{r})\,\mathrm{d}\mathbf{r} = 1. \tag{5}$$

Each particle i represents a certain volume $V_i = \frac{m_i}{\rho_i}$; this means they have mass and density. The mass m_i of each particle i is constant throughout the full simulation process. The density ρ_i needs to be evaluated at every timestep. With the substitution we get the following equa-

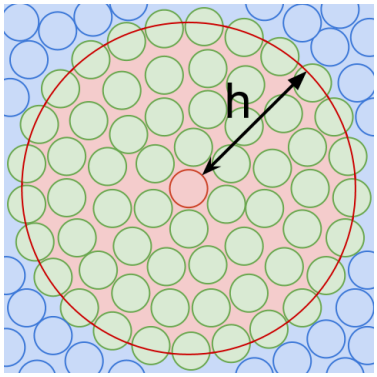


Figure 2: The smoothing distance h and the surrounding particles within it.

tion for the density at a given position $\mathbf{r}$:

$$\rho_i(\mathbf{r}) = \sum_j m_j W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{6}$$

In the SPH, the derivatives of the field need not to be evaluated. These derivatives only affect the smoothing kernels. The gradient of A is:

$$\nabla A_i(\mathbf{r_i}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h), \tag{7}$$

while the laplacian of A is given by:

$$\nabla^2 A_i(\mathbf{r_i}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h). \tag{8}$$

The main problem of the SPH is that to derive the fluid equation it is not guaranteed to satisfy all the physical principles such as symmetry of the forces and conservation of the momentum. The problems are solved via different types of smoothing kernels [13], which are discussed in the next chapters, see Fig. 3, 4, and 5.

3.3 Pressure

The applications of the SPH rule described in Eq. (1) yields:

$$\mathbf{f}_i^{pressure} = -\nabla p(\mathbf{r}_i) = -\sum_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{9}$$

As we mentioned in the previous subsection the force would not be symmetric. It can be seen simply when only two particles interact. The first particle i only uses the pressure at particle j to compute its pressure force. Since the pressures at the location of the two particles are not equal in general, the pressure forces will not be symmetric between them. Newton's 2nd law states that, to every action there is always an equal and opposite reaction: or the forces of two bodies on each other are always equal and are directed in opposite directions. There are different types of solutions for this symmetrization problem in the literature. Gross, Müller and Charypar [12] described a simple and fast solution of this problem:

$$\mathbf{f}_i^{pressure} = -\sum_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h). \tag{10}$$

Now the pressure force is symmetric because this equation uses the aritmethic mean of the pressures of the interating particles.

At first, the pressure need to be evaluated, which is done in two steps. The first is density computation from Eq. (6). Then the pressure can be computed via the ideal gas state equation:

$$p = k\rho, \tag{11}$$

where k is the ideal gas constant that depends on the temperature. Desbrun and Gascuel [10] suggest a modified version of the pressure computation:

$$p = k(\rho - \rho_0), \tag{12}$$

where ρ_0 is the rest density. This modification does not affect the pressure forces matematically. However, this makes the simulation numerically more stable, because it has influence only on the gradient field smoothed by the SPH.

3.4 Viscosity

In the SPH, the viscosity yields:

$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j}{\rho_j} \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h), \tag{13}$$

where μ is the specific viscosity constant for the fluid. The viscosity force is asymmetric, too. Since the viscosity forces are only dependent on the velocity differences and not on the absolute velocities, the solution of this problem is simple. It can be easily symmetrized by using the velocity differences:

$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{14}$$

The particle is accelerated in the direction of the relative speed of its environment.

3.5 Smoothing Kernels

The smoothing kernels used in the interpolations have a great influence on speed, stability and physical plausibility of the simulation and should be chosen wisely. Choosing a good smoothing kernel can be important for several aspects of the simulation. The numerical accuracy is highly dependent on the smoothing kernel and the research has shown that certain kernels offer better results than others [1]. The SPH uses different kernels for each calculation. Even though the Gaussian kernel has very nice mathematical properties, it is not always the best kernel to use.

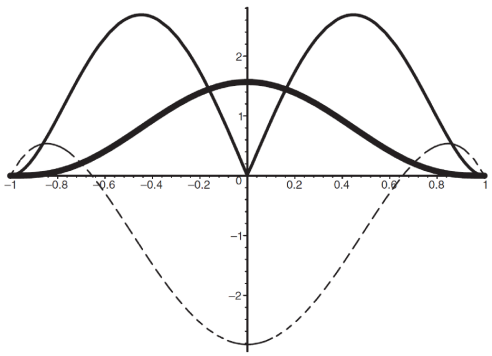


Figure 3: Smoothing kernel W_{poly6} from [12]. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian.

It does not have compact support, and requires the evaluation of computally expensive exponential function. Instead of that, the 6^{th} degree polynomial kernel can be used, which was suggested in [12] for density computation:

$$W_{poly6}(\mathbf{r},h) = \frac{315}{64\pi h^9}\left(h^2 - |\mathbf{r}|^2\right)^3 \qquad (15)$$

An important feature of this kernel is that $\mathbf{r}$ appears only in the squared form and can be evaluated without computing square root in the distance calculations.

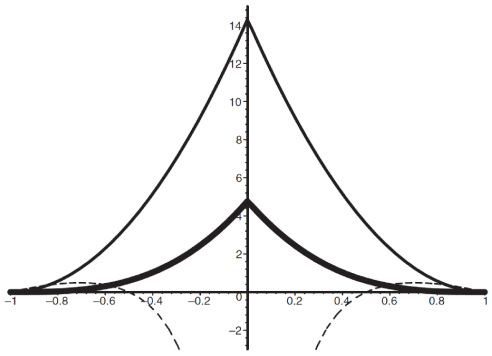


Figure 4: Smoothing kernel W_{spiky} from [12]. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian.

The gradient from the pressure field is used in the calculation of the pressure force. For our pressure force evaluations between the particles we employ the spiky kernel [12] as our pressure kernel, which yields:

$$W_{spiky}(\mathbf{r},h) = \frac{15}{\pi h^6}(h - |\mathbf{r}|)^3, \qquad (16)$$

which generates the necessary repulsion forces.

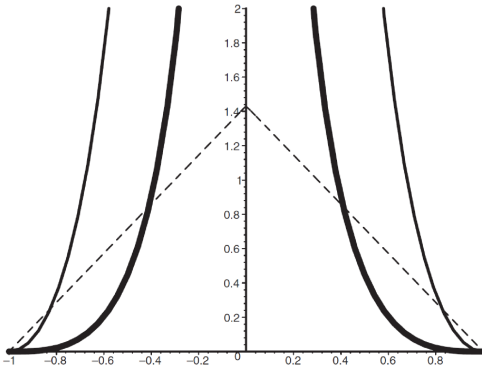


Figure 5: Smoothing kernel W_{vis} from [12]. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian.

The viscosity of a fluid is a phenomenon which is caused by the internal friction force between the particles. It decreases the kinetic energy by converting it into heat. This means that this force gives stability and smoothing effect on the velocity field in fluids. The SPH variant at the viscosity force term is:

$$W_{vis}(\mathbf{r},h) = \frac{15}{2\pi h^3}\left(-\frac{|\mathbf{r}|^3}{2h^3} + \frac{|\mathbf{r}|^2}{h^2} + \frac{h}{2|\mathbf{r}|} - 1\right) \qquad (17)$$

The Laplacian of the smoothing kernel in Eq. (17) is constrained to be positive. This is required because the forces due to viscosity to can increase the relative velocity, and thereby introduce energy and instability into the system.

How to derive the remaining equation from the previous like the gradient and the laplacian of the smoothing kernels, are discussed in [11, 16].

4 Surface Tracking and Visualization

An important part of the simulation process is rendering and visualization its results. The choice of the rendering method depends on many aspects. Two main types exist: online (real-time) and offline rendering. Offline methods provide more plausible results, but the computations take longer time. In this case, a video is created from the rendered frames. The biggest disadvantage is that the user loses the interaction with the simulation system.

4.1 Point Sprites

Rendering particles with the point sprites method is an easy, simple and fast solution comparing to the other rendering methods, like the marching cubes or raycasting method. The implementation is done in the GPU's fragment shader. Each particle is rendered as a square shaped formation. The size of this square is dependent on the distance between the particle and the camera. In the fragment shader, the pixels outside the computed radius are

discarded. Pixels that are not discarded are shaded to create the fake 3D ball effect, see Fig. 12.

4.2 Marching Cubes

Marching cubes is a method for extraction of isosurfaces in volumetric data. The method is based on the triangulation of the isosurface. The volume is sampled by a marching cube, which is traversing the volume. At each position the corners of the cube are tested, whether they are above or below the isosurface. By the configuration of the corners, triagles are generated to cover the surface.

This method highly depends on the resolution of the sampling grid, see Fig. 6. To eliminate the sharp edges, tessellation can be used. The algorithm is easily paralellizable; this means that, to speed up the rendering of the isosurface the GPU is used. The disadvantage of this method is the need of the additional normal interpolation for better shaded results.

5 Implementation

The aim of this work was to experiment with the methods described in the previous chapter and implement them running in realtime on the GPU. Next, the implementation details of simulation and rendering are described.

5.1 Uniform Grid

The uniform grid is used to track and search for the adjacent particles in the grid cells while evaluating the field quantities with the smoothing kernels. The implementation of this structure is inspired by the work of Green [17]. The space is divided into uniformly sized cells. A particle is assigned to a cell by its position. The minimal possible size of a cell can not be smaller than the size of the particles. A particle can potentially overlap several grid cells, which means that when procesing the quantities, the particles in the neighbouring cells must also be examined. The

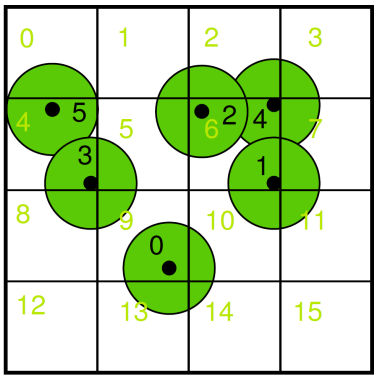


Figure 7: Particle mapping in uniform grid from [17].

assigned cell of a particle is known by its actual position, so the neighbour cells can be found, too. Each cell has its unique hash code and each particle has its unique index. Two arrays are needed: the first one contains indices of the particles and the second one contains the hash codes of the containing cells of the particles. In the next step, the two arrays are sorted using the array which contains the hash codes, as the key for sorting. This process is illustrated by figure 8. As seen in the figure, the cell with the hash code 1 contains two particles with indices 1 and 3. Finally an iteration is needed over the hash array to find the start and end positions of the cells and store them in an extra array.

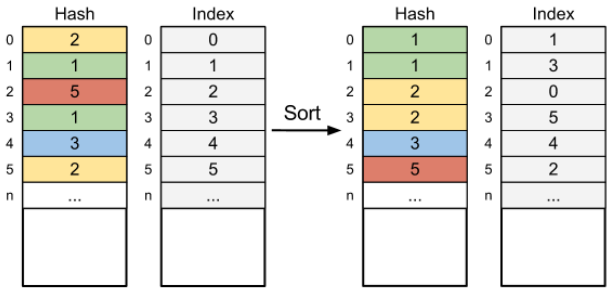


Figure 8: Sorting of particle indexes and hash codes.

5.2 Simulation

The simulation part has three main steps. The first step is density and pressure evaluation. This step is sped up by the precalculad parts of the smoothing kernels. Each kernel has a constant part without the variable vector **r**. The size of the vector represents the distance between the two examined particles. The precalculated values are stored in the constant memory of the GPU. This solution results in an extremly fast access to the values and saves computational time.

In the second step, the internal forces are evaluated, like the pressure force and the viscosity force. The details of the computations are described in the previous sections.

The last step is the integration of the velocity and further position update. At each integration step, the new velocities are computed. The velocity depends on the internal (presurre, viscosity) and external forces (computed from the particle–boundary interactions). The new position is computed from the previous position and the actual velocity. To intergrate the velocity, the Leapfrog intergration [2] is used. The name of this integrator is a result of the above formulation of it; the velocities ”leap over” the positions. The Leapfrog integration is a simple method to numerically integrate the differential equations. This method is a fairly good compromise between the naive Euler method and more advanced methods that require more than a single evaluation for each force. The default scheme can also be formulated in a form where all quantities are defined at discrete times only:

$$x_{i+1} = x_i + v_i \mathrm{d}t + \frac{a_i}{2}\mathrm{d}t \qquad (18)$$

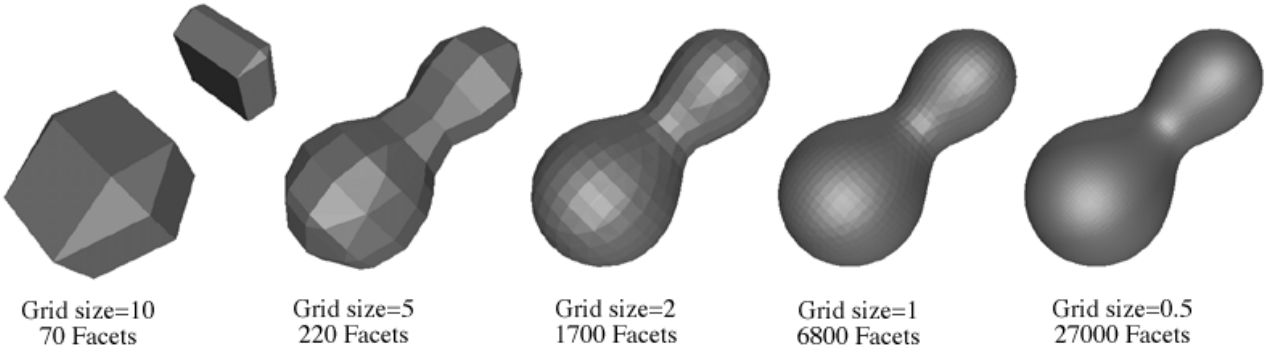


Figure 6: This figure shows the dependence of the maching cubes method on the resolution of the sampling grid from [14]. The higher grid resolution allows courser or finer approximation of the isosurface.

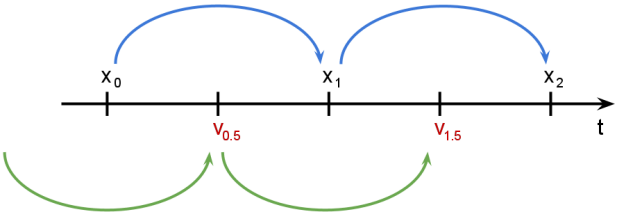


Figure 9: The leapfrog integrator.

$$v_{i+1} = v_i + \frac{a_i + a_{i+1}}{2} \mathrm{d}t. \tag{19}$$

5.3 Rendering

As we mentioned in section 4, two rendering methods were used in this work. A simpler of them is the point sprites method. This solution is computed in the GPU's fragment shader.

The second is the marching cubes method [14]. Comparing to the point sprites method, it can not be rendered direcly using the particle positions. The marching cubes method is based on the triangulation of the fluid surface. The efficient implementation of the algorithm is done by the lookup tables. Edges and corners of the cube are numbered. An 8-bit vector is used to store the configuration of the cube. Each corner has a unique number which is used as an index into the bit vector. If the corner is below the iso–value, than the nth bit (where n is the unique number of the corner) is set to 0, or to 1 when it is above the iso–value.

The lookup tables are in size of 256, because of the possible configurations (2^8). Three types of lookup tables are used. The first contains 12-bit vectors. These vectors are similar to the configuration vector of the cube, but this vector stores the intersected edges for the specified configuration. The next table contains the number of required triangles for the configurations. The third and last table contains the unique numbers of intersected edges for the required triagles.

The first step is to evaluate the occupied voxels and compute the number of the required triangles for each voxel. The following step is to count the number of the required triangles to cover the whole surface. It follows the allocation of the vertex buffer for the vertices of the triagles. In the final step the normals are interpolated. It is very important for the additional tessellation. The tessellation is used to get smoother surface. The details could be also improved by using a higher resolution grid.

6 Results

By using the GPU, the SPH method, and the marching cubes method, the goal of this work was achieved. The SPH method is running in real-time with 60000 particles when no rendering method is used. When using lower grid resolutions, there is a performace drop compared to higher resolutions, see tables 1, 2, 3, and 4. This effect is caused by the high density of particles in the cells. Per cell computations take longer time, because of the higher particle count. The test results of the implemented simulation and the rendering methods are summarized in tables 1, 2, 3, and 4.

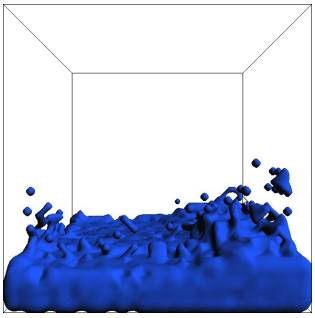


Figure 10: Fluid rendered with the marching cubes method.

When using the marching cubes and/or the tessellation there is a performace drop at higher grid resolutions, see the test results.

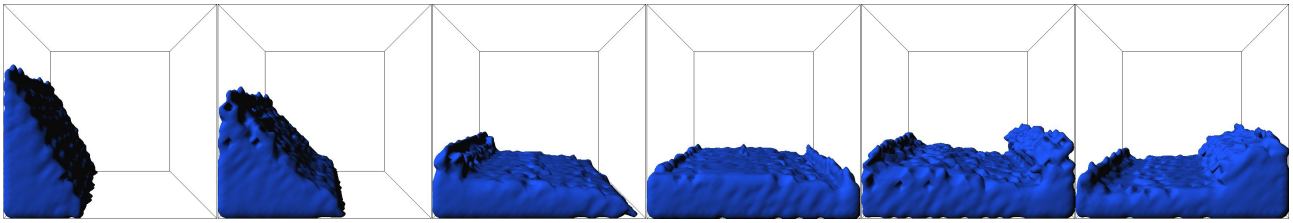


Figure 11: The fluid motion is simulated by 15,000 fluid particles with marching cubes rendering.

	Number of particles: 15000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	6	5	4
MC	2	5	12
NI	8	11	15
Sum	16	21	31

Table 1: The benchmark shows the results measured in milliseconds at different grid resolutions. *SIM* is the simulation, *MC* is the Marching cubes rendering, and *NI* stands for the normal interpolation. In each table, the upper row contains the number of particles used in the benchmark.

	Number of particles: 30000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	19	13	9
MC	2	5	12
NI	9	12	18
Sum	30	30	39

Table 2: Benchmark results for 30,000 particles.

The algorithms are implemented in the C++ programming language. The sourcecodes for the GPU were implemented for the CUDA runtime API 4.0. The API allows high abstraction level and provides fine support for programming the GPU. The final program was tested on the NVidia GeForce 540M graphics card, with 96 cores.

7 Conclusions and Future Work

The SPH method is a very realistic and fast appoximation of the real world fluids, but it has some limitations. One of these limitations is the need of large number of particles for the simulation to get realistic results.

Many experiments were made in this work to speed up the SPH and its rendering methods. The uniform grid helps the access to adjacent particles. This structure speeds up the simulation, but it needs additional computations, which can be better optimized, like the radix sorting of its arrays.

In the future more experiments can be done to speed up the searching and sorting the particles. The additional surface tension computation can improve the surface details of the fluid and make it more realistic. Other experiments can be done to improve the existing algorithms of rendering, e.g. extract a better approximation of the surface with marching cubes or implement the raycasting method, which provides much more plausible results.

	Number of particles: 45000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	31	22	18
MC	3	5	13
NI	9	12	20
Sum	43	39	51

Table 3: Benchmark results for 45,000 particles.

	Number of particles: 60000		
Grid res.	30x30x30	50x50x50	80x80x80
SIM	45	32	24
MC	3	5	13
NI	10	13	30
Sum	58	60	67

Table 4: Benchmark results for 60,000 particles.

References

[1] A. J. C. Crespo. *Application of the Smoothed Particle Hydrodynamics model SPHysics to free-surface hydrodynamics*. PhD thesis, University of Vigo, 2008.

[2] D. H. Eberly. *Game Physics*. Elsevier Science Inc., 2003.

[3] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[4] R.A. Granger. *Fluid Mechanics*. Courier Dover Publications, 1995. ISBN 0486683567.

[5] M. J. Harris. Fast fluid dynamics simulations on the gpu. In *In GPU Gems, Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addision-Wesley, 2004.

[6] J. J. Monaghan. Smoothed Particle Hydrodynamics. *Reports on Progress of Physics*, 68:8, 2005.

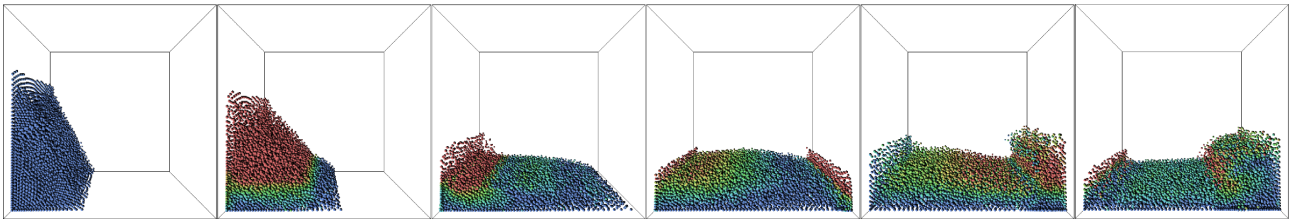


Figure 12: The fluid motion is simulated by 15,000 fluid particles with point sprites rendering. Colors indicate the velocity field of particles, red is high and blue is the low.

[7] R. Westermann J. Krüger. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization*, 2003.

[8] K. Erleben, J. Sporring, K. Henriksen, H. Dohlmann. *Physics Based Animation*. Charles River Media, 2005.

[9] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, 1977.

[10] M. Desbrun, M. Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76, 1996.

[11] M. Kelager. Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics, 2006.

[12] M. Gross M. Müller, D. Charypar. Particle-based fluid simulation for interactive applications. In *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation*, 2003.

[13] O. E. Krog. *GPU-based Real-Time Snow Avalanche Simulations*. PhD thesis, Norwegian University of Science and Technology, 2010.

[14] P. Bourke. Polygonising a scalar field [online]. `http://paulbourke.net/geometry/polygonise/`, 1994.

[15] J. J. Monaghan R. A. Gingold. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181:375–389, 1977.

[16] S. Auer. *Realtime particle-based fluid simulation*. PhD thesis, Technische Universität München, 2009.

[17] S. Green. Particle Simulation using CUDA [online]. NVidia Corporation, 2010.

[18] V. Springel. Smoothed particle hydrodynamics in astrophysics. *Annual Review of Astronomy and Astrophysics*, 48:391–430, 2010.

[19] J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999.

[20] Y. Uralsky. Practical metaballs and implicit surfaces. In *Game Developers Conference 2006 Presentation*, 2006.

[21] H. E. Cline W. E. Lorensen. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987.

Gaze-dependent Ambient Occlusion

Sebastian Janus*
Supervised by: Radosław Mantiuk†

Faculty of Computer Science
West-Pomeranian University of Technology
Szczecin / Poland

Abstract

Ambient Occlusion is a method of creating shades on the scene, due to occlusion. It is a good looking approximation of the light radiation, however it is very expensive method. It needs a large number of samples to get fair effects. In this article we propose a speed increase of the AO rendering, by using the eye tracker. Human cannot see high frequency details in parafoveal, and we can render this area with less accuracy. We decrease the number of AO samples with distance from the observer gaze point. The absence of AO shading in parafoveal is being rarely noticed and reducing the samples gives us considerable rendering speed boost.

Keywords: Ambient Occlusion, Eye tracking, Modern computer graphics

1 Introduction

Ambient Occlusion (AO) is a shading algorithm, which adds a reality to the rendered scene. It approximate how the given point is occluded by other objects (surfaces). However, it is fair complex and still it is hard to achieve a real time AO performance on the nowadays GPUs. In this work we provide a solution that speeds-up the AO rendering without significant decrease of the quality of the final image.

We present a concept of rendering ambient occlusion self-shadows affected by the information about the humans' viewing direction and its limited region of interest (ROI). Therefore, if we knew the point on which observer has focus, we could render this point surroundings with maximum precision and further regions with minor quality. We use the gaze-dependent Contrast Sensitivity Function to alter the influence of the AO factor and to model decrease of rendering quality.

During experimental evaluation we evaluate whether the humans are capable of seeing the difference of the rendering quality outside the ROI.

To achieve the interactive rendering, we base our AO implementation on nVidia OptiX[1] library, which operates on CUDA[2] for the supreme speed of the complicated calculations. It computes ambient occlusion factors with differential accuracy, depending on the location of human gaze point captured by the eye tracker.

In Section 2 we describe the Ambient Occlusion algorithm and discuss why the full Ambient Occlusion method has been chosen instead of cheaper approximated methods. Then in Section 3 we present our concept of the gaze-dependent ambient occlusion rendering. In Section 4 we describe the implementation. The results are discussed in Section 5, followed by conclusions and future work in Section 6.

2 Background

Ambient occlusion is a shading model which is used to increase scene realism in rendering systems based on the local illumination models. It requires much less computation in comparison to the full global illumination solutions, however it still needs demanding resources to achieve high quality renderings.

2.1 Ambient Occlusion

In Phong reflection model, diffuse and specular reflections are varying due to observer and lights position, but ambient reflections are constant. Having these assumptions we miss the self shadows of the rendered objects - which is a big lack of reality. Adding Ambient Occlusion [2] algorithm for computing the ambient reflections factor creates very convincing soft shadows, which combined with indirect lighting gives realistic results. The model does not look flat - what often is happening with indirect lighting and multiple light sources. The result looks similar to Global Illumination and it is possible to say that it simulates it. Good point is, it is of course less complex than full global illumination [1].

This method is an integration of visibility, computed from each pixel on the rendering screen. This integration is solved by Monte Carlo method, where we achieve the

*sebastian.janus@o2.pl
†rmantiuk@wi.zut.edu.pl

[1] See: http://www.nvidia.com/object/optix.html
[2] See: http://www.nvidia.com/object/cuda_home_new.html

result by casting very large number of samples - counted in hundreds. Sample rays are traced from every point into random directions on the hemisphere:

$$k_a = \frac{1}{\pi} \int_{\Omega} V_p(\vec{\omega})(N \cdot \vec{\omega}) d\omega, \quad (1)$$

where k_a denotes the occlusion factor. V stands for the binary visibility function from the certain point p, which returns positive values when ray does not intersect any geometry until reaching some given distance (for the purpose of ray tracing in closed scenes), and a negative value when traced ray hits any object. p and its normal vector N define the surrounding hemisphere Ω [5].

The k_a factor is used in the Phong's reflection equation:

$$I_p = k_a i_a + \sum_{\text{m} \in \text{lights}} (k_d(\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s(\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s}), \quad (2)$$

where i_d and i_s are defined as the intensities of the current light source, i_a is a constant value or a sum of ambient light of all light sources, k_d is a diffuse reflection constant (lambertian reflectance, depending on the angle between the direction $\hat{L}_m$ toward current light source and the surface normal vector $\hat{N}$), and k_s is a specular reflection constant with $\hat{R}_m$ being a light perfectly reflected ray.

The calculation of the ambient occlusion factor is depicted by the Algorithm 1.

Algorithm 1 Ambient occlusion algorithm

```
for i := 1 → screenWidth do
  i ← i + 1
  for j := 1 → screenHeight do
    j ← j + 1
    {Calculate ambient occlusion for every pixel}
    occlusion:=0;
    for k := 1 → aoRaysNumber do
      k ← k + 1
      {Cast rays in a random directions from the following pixel};
      occlusion+ = castAoRay(randomDirection)
      { castAoRay returns 1 if it hits something}
    end for
    occlusion = occlusion/aoRaysNumber
    { Calculate the ambient occlusion factor. }
  end for
end for
```

2.2 ScreenSpace Ambient Occlusion

The idea of the Screen-Space Ambient Occlusion (SSAO) was proposed by Crytek game studio in their Crysis game as the fast alternative of the standard ambient occlusion technique [4]. SSAO works in real time, although renders effects of lower quality. The idea lies on using Z-buffer data to compute visibility function for every pixel. It takes a pixel surrounding points and analysis their Z value [4].

However, this method has some important disadvantages, starting with self-occlusions. SSAO samples are taken from the inside of a sphere around each pixel - in non-occluded surfaces almost 50% comparisons return 'occluded' result. This causes haloing around the objects - where the self-occlusion effect disappears. These halos are visible around the boxes in Figure 1. There are various improvements to the original SSAO algorithms trying to fix that haloing, but they are not universal - they simply do not work for all cases.

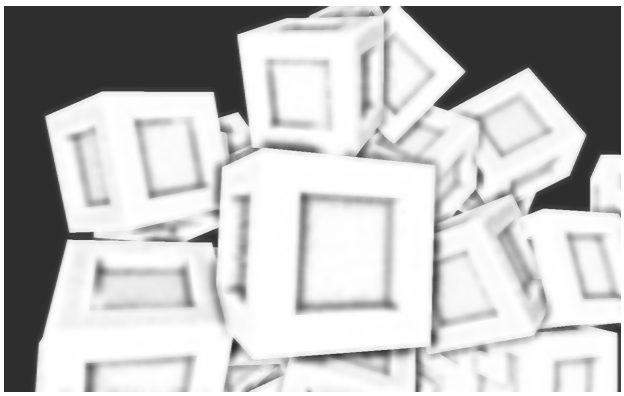


Figure 1: Inaccuracy of the SSAO algorithm may cause black and white halos on the screen [6].

This method is not precise because of the small samples number and the idea of the algorithm - it produces some noise [9]. Moreover, the only objects taken into the consideration are the visible ones. Everything outside the frustum is not interfering with rendered scene. Performance is also depending on the scene - closer objects will require larger radius of sampling.

All these cons made us to the decision for favouring the standard ambient occlusion technique. The main problem is that it requires a lot of computations - for every pixel one should trace hundreds of rays to make satisfactory results. For example for 1680x1050 image resolution and 500 AO factor test rays, the AO algorithm requires tracing of 882 million rays to render one frame. For the full HD (1920x1080) and increased quality to 1000 rays, the sum rises to 2,037 millions what makes the AO not possible to render in real time based on the contemporary graphics systems.

3 Gaze-dependent rendering of Ambient Occlusion

Gaze-dependent ambient occlusion is a solution of providing a full detailed ambient occlusion effect in a certain region of interest. The further from the gaze point, the less detailed ambient factor is rendered, saving the computing time and leaving observer with a feeling that the scene is fully detailed.

3.1 Rendering system

The main goal of this work is to track observers gaze, and to use information about the gaze direction to render parafoveal parts of the screen with less quality. The outline of the gaze-dependent ambient occlusion system is presented in Figure 2.

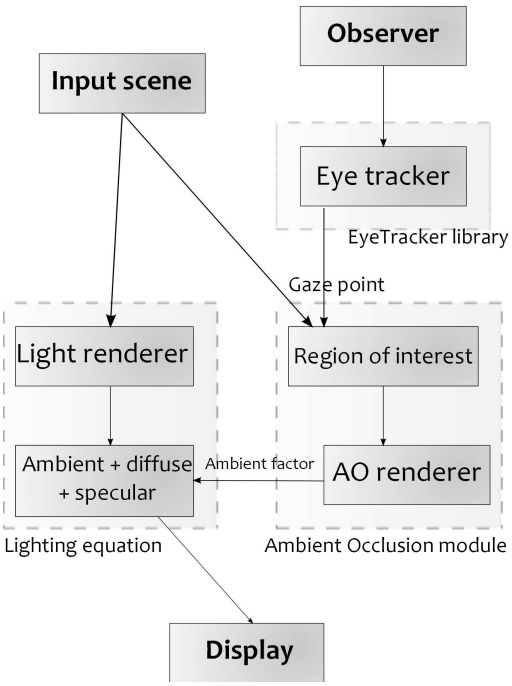


Figure 2: Gaze-dependent ambient occlusion rendering system.

The input data is a 3D scene defined by *.obj file. We have also an observer, whose eyes are tracked. From the eye tracker library we receive the actual gaze points, which are used to calculate the ROI shape and position. Then we have the main lighting renderer, calculating final colour from the Phong lighting equation, and AO renderer, calculating the ambient factor with use of the gaze-dependent ambient occlusion algorithm. Finally, we blend these two ambient factors depending on the distance from the centre of the ROI and we display it.

3.2 Region of interest sampling

Changing the rendering scene by modifying the ambient occlusion factor is rather subtle and it is treated as high frequency information. Therefore, we can use a gaze dependent Contrast Sensitivity Function (CSF) for a function describing how the human eye treats contrast changes in a given distance from a gaze point [15].

We will use drop-off of visual sensitivity across the visual field, for modelling the AO accuracy. We decrease ambient occlusion for the constant ambient factor in lighting equation while we increase the distance from the ROI

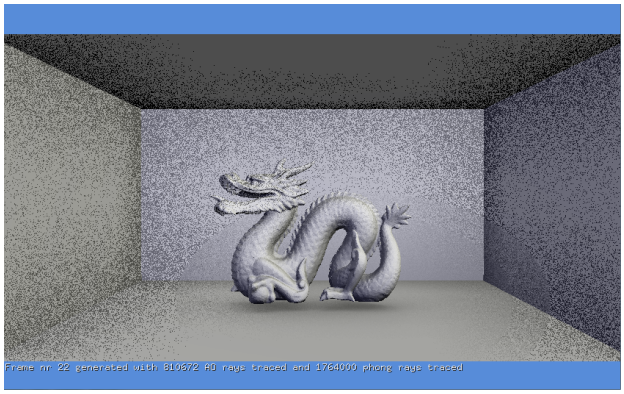


Figure 3: Gaze-dependent ambient occlusion without blending (visible noise).

centre. In the meantime, we decrease the number of the AO sampling rays. We do not need high accuracy, when the impact of this factor is getting more and more inconsiderable [3].

In addition, with CSF we avoid visible noise (see Figure 3), which is adverse for the observer. It is noticeable even in the parafoveal vision and it generates temporal aliasing. The noise is produced by the ambient occlusion when we use small samples number. In our work, that noisy result is hidden by blending with the normal lighting equation ambient factor.

The precision downfall is given by the contrast sensitivity function:

$$C_t(E,f) = C_t(0,f) * exp(kfE), \quad (3)$$

where C_t denotes contrast sensitivity for spatial frequency f at an eccentricity E, k determines how fast sensitivity drops off with eccentricity (the k value is ranged from 0.030 to 0.057). Based on the above equation, the cut-off spatial frequency f_c can be modelled as:

$$f_c = min(max\_display\_cpd, 43.1 * E_2/(E_2+E)), \quad (4)$$

with $E_2 = 3.118$, which is retinal eccentricity at which the spatial frequency cut-off drops to half its foveal maximum (from 43.1 cpd to 21.55 cpd, see details in [16]). In this equation we flatten the contrast sensitivity with $min(max\_display\_cpd, ...)$ operator to take into consideration the limited resolution of our display (see Section 5.1). That flattening is represented on the Figure 4 as a magenta line.

The plot of the contrast sensitivity function is presented on the Figure 4. As we can see, it is quickly decreasing and then staying around some low level. On the Figure 5 we can see a region of interest mask preview. The lighter areas mean that AO is computed with maximum precision, the darker areas mean lower precision. Blending factors of AO ambient and ambient from Phong lighting equation

have the same distribution - the lighter areas means higher weight of the AO ambient factor.

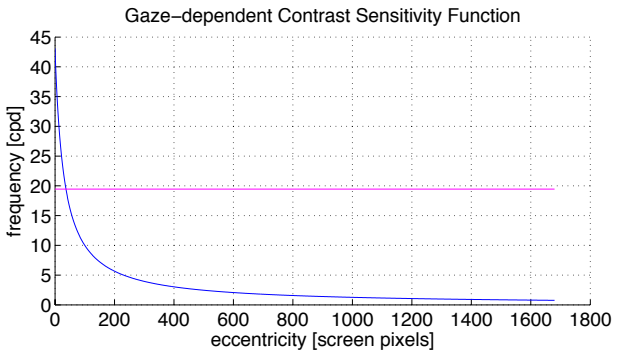


Figure 4: Gaze-dependent Contrast Sensitivity Function. The magenta line denotes threshold frequency of the display used in the experiments.

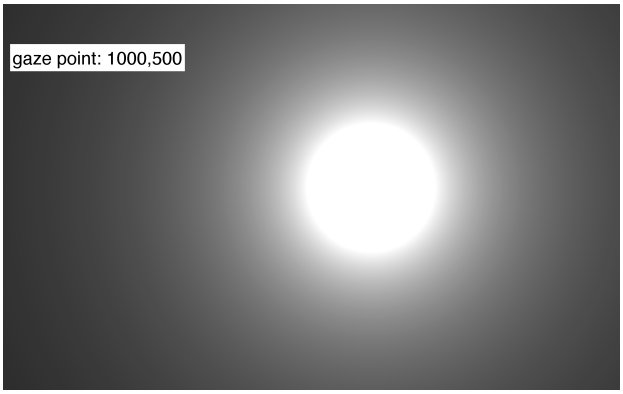


Figure 5: ROI based on Contrast Sensitivity Function preview.

3.3 Eye tracker fixations

Eye trackers capture two types of eye movement called *saccades* and *smooth pursuit*. A smooth pursuit is active when eyes track moving target and are capable of matching its velocity. A saccade represents a rapid eye movement used to reposition the fovea to a new location, which lasts from 10 ms to 100 ms [12]. The main goal of the gaze tracking is to capture a single location an observer intents to look at. This process is known as a *visual fixation*. A point of fixation can be estimated as a location where saccades remain stable in space and time [13]. We approximate this mechanism by averaging a number of raw gaze points. There are many fixation algorithms which are trying to provide a gaze point from these samples, however they are complicated and none of them had worked better for our solution, where we expect a stable results, than simple average of points [14].

Additional stabilisation of the gaze position is achieved by rendering delay. There are about 5-10 frames per second rendered, and we take gaze points array once before rendering new frame. It means that we are averaging gaze samples from about 100-200 ms period, which effects with a lot of samples and in the outcome the mean value is not affected by few different samples. In that case it is a fair advantage.

Human field of view for one eye is about 130° in horizontal plane and about 120° in vertical plane. What is more, the second eye is extending the horizontal plane to about 200°. The binocular field of view then is about 60° [10]. Assuming that, for human angle of view the ratio between horizontal and vertical planes is like 1.538 : 1. That is why our region of interest shape would not be a simple circle, but a widened ellipse (to take advantage from the differing field sizes). This would not work if we will look with single eye, but for binocular gaze it will work perfect.

4 Implementation

In this section we provide a description of our implementation of the gaze-dependent AO system, proceeded by a short introduction to the OptiX library. Our application is based on the AO sample from the OptiX SDK package. We extended this sample to the gaze-dependent technique supported by the eye tracker library.

4.1 OptiX

Our software is implemented using OptiX engine [11]. It is developed by nVidia and it is described as programmable ray tracing framework, which can be used to rapidly build ray tracing applications. Computing speed of application based on that engine gives fast results across nVidia GPUs with conventional C or C++ programming.

OptiX is helpful in detecting collisions, calculating sound volume, radiation research, and other rendering purposes. Developers can write their own single ray programs. These programs are divided into few categories: closest hit, any hit, intersection, selection, ray generation, miss and exception programs. Using an ensemble of these programs gives us a rendering algorithm. Shading language is based on C/C++ for CUDA, with all its features like pointers, templates, and overloading. One is encouraged to use object model as well.

4.2 Ambient occlusion based on OptiX

Ambient occlusion based on OptiX is divided into a host program (written in C++), and a few GPU shaders programs:

- Ray generation program - responsible for proper generation of the rays from the viewer towards the scene.

Whenever the camera has moved, different rays are generated.

- Closest hit program - responsible for calculating lighting radiance. This is main shader used for tracing rays from the viewer - it looks for intersections with scene objects, and if it finds any it starts computing the colour basing on the Phong lighting equation and our gaze-dependent AO.
- Any hit program - responsible for calculating any hit occlusion. It is used for tracing rays from the intersection points (pointed by the rays which are calculating radiance). We can call them secondary rays, which are cast in a big number to obtain the ratio for hit/miss rays - it gives us the ambient occlusion factor.
- Miss program - responsible for determining the background colour - it is used when our primary rays do not hit anything.
- Exception program - responsible for exceptions - used in case we get incorrect value of lighting.

Apart from the host program which is responsible for all the input/output, communication with user and setting (or changing) the parameters of the GPU programs, the main code is in the closest hit program. There we are calculating the number of the ambient occlusion rays to be cast, there we cast these rays, and there we finally calculate the colour of each pixel.

As it was said before our application is based on a ambient occlusion sample. The main differences instead of main host program (another scene, controls and so on) are in the closest hit shader. We had to provide ROI computation based on contrast sensitivity function, which gives us varying ambient occlusion rays number. Because of that information which pixel are we computing at the moment was significant. Then we had to implement blending the AO ambient factor and Phong lighting equation ambient factor, and there comes gaze-dependent ambient occlusion.

4.3 EyeTracker library

We use the ETlib library for the purpose of tracking observers' gaze direction. This software is responsible for managing communication between the eye tracker software (which runs on another computer) and our application. The eye tracking session starts with the calibration - observer looks at on given point on the screen. The process is finished after registration of 5 points. Precise calibration is extremely important because it affects further accuracy of captured data [7, 8].

Using ETlib we can receive last gaze point (which is not stable - unfortunately, human eye is moving many times in a very short period) or a set of gaze points since the last request. We use the second approach, and with an array of points we make an average point - then we proceed this point as a variable which will be used then by the GPU programs.

5 Experimental evaluation

Our objective is to present, that we can minimise the ambient occlusion sampling in parafoveal. What is more it gives us a performance boost. We present a images for different ROI position, and analyse rendering times.

5.1 Stimuli and hardware setup

The scene presents the fixed Stanford Dragon Model[3], enclosed in the 5 walls box. The scene consist of 50,008 vertices and 100,005 faces, and gives us good performance test object. We render this scene with full frame ambient occlusion to obtain an ideal image, or we render it with use of ROI (controlled by eye tracker) using gaze-dependent AO.

In our experiment, we used the SMI RED250 eye tracker, which gives us refresh rate 250 Hz and accuracy 0.5°. The computer is equipped with 2.8 GHz Intel i7 930 CPU with 8 GB of RAM, Windows 7 64bit OS, and a GPU nVidia GeForce 480 GTI 512MB - one of the fastest nVidia graphic cards. The hardware setup is presented in Figure 6.

Figure 6: Apparatus used during evaluation and experiments. The RED250 eye tracker is located under the display screen.

The lab display is 22 inch Dell, with the 1680x1050 resolution (60 Hz). It is measuring 47.5 cm wide and 30 cm high, what gives 43° horizontal, and 28° vertical. With the screen resolution 1680x1050 that gives 40 pixels for one degree, which is about 20 cpd. That is our maximum,

[3] http://www.mrbluesummers.com/3572/downloads/stanford-dragon-model

which is marked as *max_display_cpd* in Equation 4 and as a magenta line in Figure 4.

Eye tracker is connected to the remote computer. We launch it using remote desktop, and the ETlib (See subsection 4.3) gives as the array of gaze points since last call.

5.2 Results

Quality of rendering

An example rendering with the full frame ambient occlusion is presented in the Figure 7. We consider the full frame rendering with 400 AO rays per pixel as image in all quality comparisons.

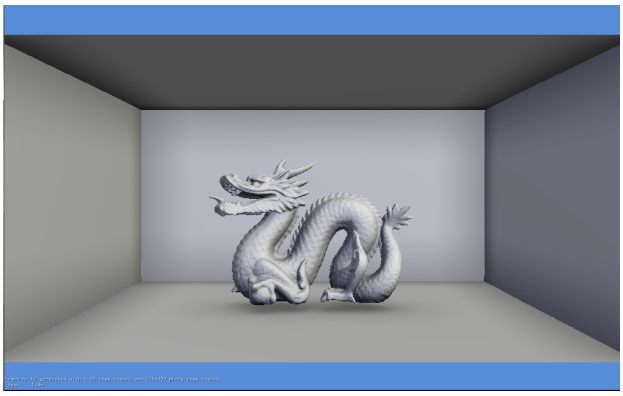


Figure 7: Ideal, reference image with full frame ambient occlusion

In Figure 8 one can see images rendered with gaze-dependent ambient occlusion for various locations of the ROI. Please, notice that shading caused by the AO factor is stronger in the centre of the ROI and weakens with the distance.

On the upper image in Figure 8 there is visible AO effect in the upper corner of the box, and there is no shadow below the dragon model. On the image where the position of the ROI is on the dragon these shadow is visible very well, and there are little shadow on the wall behind the dragon. There are no shadows on the higher corners at all.

During the pilot study with the eye tracker, we assessed the quality of the AO shadows as a very good in comparison to the reference image. The contrast sensitivity function was doing well, and smaller shades in a greater distance from a centre of ROI were rarely noticeable.

Rendering time

To compare timings for the full frame AO and the gaze-dependent AO we measure the speed of rendering for three different camera settings (called as Camera 1, 2, and 3, see Figure 9).

We achieved 1.26 fps, 0.96 fps, and 0.62 fps for Camera 1, 2, and 3 respectively for the resolution of 840x525

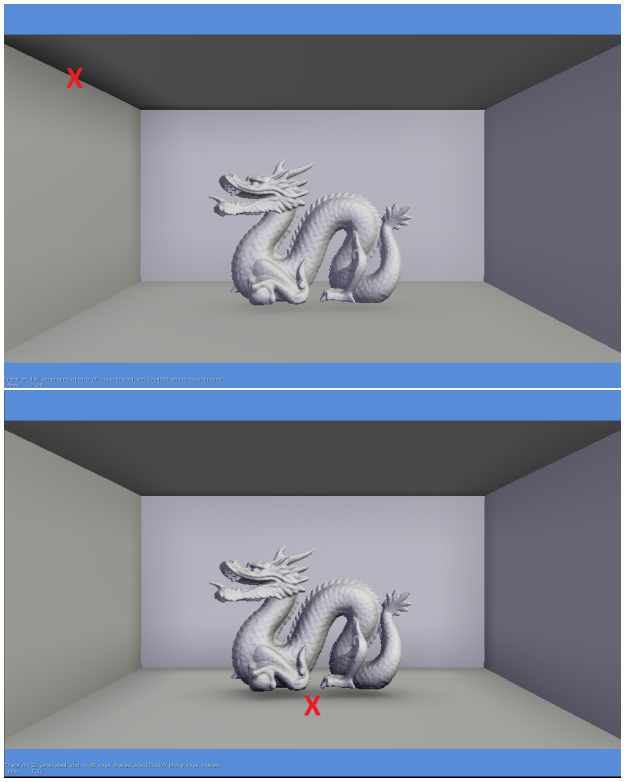


Figure 8: Image rendered with gaze-dependent ambient occlusion. The red X points the centre of the ROI.

pixels and 400 AO rays (the resolution was reduced for performance reasons).

The same images was rendered using eye tracker and the gaze-dependent AO rendering technique. The rendering speed depends on location of the ROI. There are more computations (e.g. intersection tests) in regions of the scene with more triangles so AO rendering time increases. However, the overall rendering time using the gaze-dependent technique is shorter in comparison with the full screen AO, even by 276% in the best case (see Table 1).

Generally, the results show significant performance increase in a regions of a small triangles number (box walls all around the screen). Worse outcome is when we take the hardest to compute environment - middle of the screen (the biggest samples number) and the dragon object (high level of the triangles - ray tracer has harder work). However, even in that worse case, we get a noticeable rendering speed-up.

5.3 Discussion

Using gaze-dependent ambient occlusion can increase rendering speed without a noticeable quality loss. Better GPU

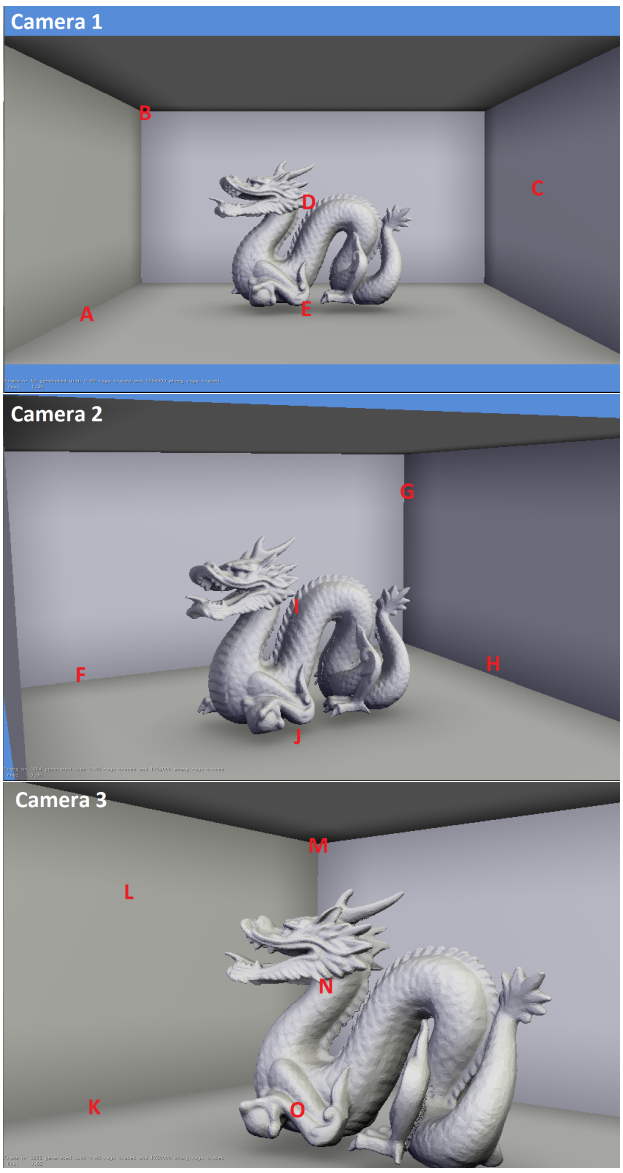


Figure 9: Locations of the ROIs depicted as the red letters.

would be useful, both for rendering smooth ambient occlusion and for performing study concerning rating a visibility of AO in the parafoveal area. The faster reaction time would also improve the gaze-dependent approach in context of fast eyes movement.

For the time of writing this article, we did not use the newest graphic card (nVidia GeForce GTX 580), which has about 20% faster memory bandwidth, texel rate and pixel rate. Probably, using two GPUs connected with SLI[4], would give us better times. With such hardware improvements, we would be closer to the smooth real time rendering and the results would be even better.

[4]SLI stands for Scalable Link Interface, which is the name of technology allowing to link two or more GPUs to perform parallel processing of a computer graphic for single output.

Table 1: Rendering speeds for gaze-dependent AO. Speed-up is $((FPS - OriginalFPS)/OriginalFPS) * 100\%$, where $OriginalFPS$ means the rendering speed of full frame rendering.

Camera	Gaze-point	Rendering speed	Speed-up
Setting 1	A	4.10 fps	201%
	B	4.21 fps	210%
	C	4.06 fps	199%
	D	1.68 fps	24%
	E	2.11 fps	55%
Setting 2	F	2.36 fps	146%
	G	2.65 fps	176%
	H	2.19 fps	128%
	I	1.30 fps	35%
	J	1.40 fps	46%
Setting 3	K	1.82 fps	194%
	L	2.33 fps	276%
	M	2.24 fps	261%
	N	1.00 fps	61%
	O	1.17 fps	89%

6 Conclusions and future work

Summarising, we are capable of having significant rendering speed increase with the Ambient Occlusion shading. Furthermore, the result of rendering worse shaded image in the parafoveal is being skipped by the human eye.

The impression of the gaze-dependent image is received as a bit worse, but mainly because of not perfect eye tracking. Sometimes, the ROI is quickly moving which could be uncomfortable for the viewer, especially when suddenly the gaze point escapes from the real gaze point for one animation frame. However, there is a need to perform a experiments on a frequent group of people to know in which way should we improve our approach. Anyhow, with improvement of the fixation algorithm we could achieve better results. Also, with the improvement of the hardware we will have smoother animations.

References

[1] P. Berto. Occlusion tutorial. 2007.

[2] M. Bunnell. *GPU Gems 2, Chapter 14, Dynamic Ambient Occlusion and Indirect Lighting*. Addison Wesley, 2005.

[3] J. Yang E. Peli and R. B. Goldstein. *Image invariance with changes in size: the role of peripheral contrast thesholds*. JOSA A, Vol.8, Issue 11.

[4] M. Mittring14 Crytek GmbH. Finding next gen cryengine 2. 2007.

[5] S. Hill. Hardware accelerating art production. 2004.

[6] M. Lagergren. Ssao. 2009.

[7] R. Mantiuk, M. Kowalik, A. Nowosielski, and B. Bazyluk. Do-it-yourself eye tracker: Low-cost pupil-based eye tracker for computer graphics applications. *Lecture Notes in Computer Science (Proc. of MMM 2012)*, 7131:115–125, 2012.

[8] R. Mantiuk, A. Tomaszewska, and B. Bazyluk. Gaze-dependent depth-of-field effect rendering in virtual environments. *Lecture Notes in Computer Science (Proc. of SGDA 2011)*, 6944:1–12, 2011.

[9] J. M. Mendez. A simple and practical approach to ssao. 2010.

[10] M. H. Nizankowska. *Podstawy okulistyki*. VOLUMED, 1992.

[11] S. Parker. Interactive ray tracing with the nvidia optix engine, 2009.

[12] D. A. Robinson. The mechanics of human saccadic eye movement. *Journal of Physiology*, 174:245–264, 1964.

[13] D. D. Salvucci and J. H. Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications (ETRA)*, pages 71–78, New York, 2000.

[14] Frederick Shic, Brian Scassellati, and Katarzyna Chawarska. The incomplete fixation measure. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, ETRA '08, pages 111–114, New York, NY, USA, 2008. ACM.

[15] P. Wenderoth. The contrast sensitivity function.

[16] Qi X. Makous W. Yang, J. *Zero frequency masking and a model of contrast sensitivity.* Vision Research.

Visualization

Partial Volume Effect Correction on the GPU

Zsolt Márta*
Supervised by: Dr. László Szirmay-Kalos†

Department of Control Engineering and Information Technology, Budapest University of Technology (http://cg.iit.bme.hu)

Abstract

This paper proposes a Partial Volume Effect correction algorithm for PET/CTs, which improves the PET (Positron Emission Tomography) data based on the registered and segmented CT (X-ray Computed Tomography) volume. The algorithm is based on mathematical morphology operations and identifies those regions where both the PET and the CT data have boundaries. The correction is restricted to these regions thus we can avoid the migration of artifacts and boundaries present in the CT but irrelevant for the positron density. Thus, the algorithm also maintains activity correctness. The proposed method is implemented in a massively parallel framework.

Keywords: Partial Volume Effect, GPU, Image Processing, PET

1 Introduction

Positron Emission Tomography allows functional analysis of physiological processes. However, its poor spatial resolution limits accurate quantitative measurements particularly in small structures partially occupying the *point-spread function* (*PSF*) of the scanner [7]. The effect caused by this poor spatial resolution is commonly known as the *Partial Volume Effect* (*PVE*).

This paper presents an approach for PVE correction using registered and segmented CT images. Utilizing the high resolution anatomical CT image, our method enhances region borders in PET images based on the assumption that region boundaries present in both images are co-located.

2 Background

PVE actually refers to two phenomena that affect image intensities, the blurring effect and sampling error of the finite discrete grid. The sampling error is due to the finite voxel size, so multiple tissue types are included in most voxels. The blurring effect and the poor resolution of PET imaging comes from detector design and physical properties of the methodology used. Physical reasons of image blurring include time and space offset of annihilation and decay origin (called positron range), acollinearity of the two generated photons, photon scattering (both in tissues and detector crystals), finite size of crystal detectors, and the deficiencies of the detector electronics [5]. Mathematically, the PET image is the result of a low-pass filter, the PSF of the scanner, with some reconstruction noise added.

PVE severely affects tissue boundaries as blurring reduces high-frequency details. This results in the underestimation of uptake values in regions of interest (ROIs). More specifically, regions with higher activity are distorted in a way that their total activity is spread across a large area. Due to reasons earlier described, PVE does not change the total activity in regions, only spreads them, possibly leading to lower maximum values in smaller areas. This spreading results in a *spillover* effect, i.e. surrounding tissues with lower activity seem to have gained uptake. The smaller the region is, the dimmer it looks resulting in erroneous qualitative assessment especially with small lesions [9].

2.1 Partial Volume Effect correction

Due to its high medical relevance, significant efforts have been made to correct PVE, although there is no generally accepted method to date. Partial volume correction (PVC) methods were first examined for brain scans, due to the numerous small structures it contains. Many algorithms deal with the problem on *regional level*, resulting in regional uptake values instead of corrected images. In addition, often *a priori* information is required, for example, the PSF for deconvolution methods, or the assumption that the original activity is constant in specific regions. The PSF, however, is usually not available and may depend on the measured object as well, which restricts the availability of this approach.

Another family of algorithms deals with the problem on *pixel/voxel level*. These produce PVE corrected images enabling visual evaluation. Particularly popular pixel level methods are based on *image fusion* techniques, which exploit simultaneous and automatically registered PET and CT acquisition and fuse functional and anatomical information. The main assumption is that tissue boundaries

*mzsolt90@gmail.com
†szirmay@iit.bme.hu

appearing in both images enables correction relying on anatomical data. This assumption seems realistic as different tissue types have different density (thus different intensity in CT scan), and also the radiotracer density depends on the tissue type. Therefore on the boundaries of different tissues, it is assumed that both CT and PET values change and thus CT and PET boundaries are co-located.

In 2006, Boussion *et al.* [1] published a novel multiresolution approach based on wavelet transform and image fusion. Their method extracts the high-resolution components from the anatomical image, and incorporates in the low-resolution PET image using a simple intensity scaling transform between CT and PET wavelet coefficients [8, 2]. In their paper, Boussin *et al.* used a simple pixel-by-pixel division to find the appropriate global scaling factor. This produces uneven edges due to PET fluctuations, and introduces CT noise in addition to PET noise. Moreover, in homogeneous anatomical areas where high frequency components are near zero it results in extreme corrected values.

Global scaling ignores local image features and when CT and PET activities are less correlated, it causes visual artifacts and quantitative errors at boundaries. Since the correction is a function of global PET features and local CT features only, it inherently cannot adapt to local dissimilarities between PET and CT intensity. A specific issue is when the functional image already contains high-frequency information and additional values are added leading to hyperintensive edges in corrected images as shown in Figure 1. Furthermore, extreme values affect the whole image as global scale calculations involve them as well. On the other side, low-intensity features become less significant as they weigh little in averages. Moreover, it may happen that CT and PET intensities are negatively correlated in some part of the image, thus the method amplifies PVE instead of correcting it.

Another possible approach may involve using local scales. For each point the scaling between wavelet coefficients is determined by a function of nearby wavelet values. In this case, the window size greatly affects the results: using a too small window results in too little information to define scaling and thus leads to a visually and quantitatively incorrect image due to PET fluctuations and extremes. Therefore the correction becomes unstable. On the other hand, using larger window blurs local image features. This window size parameter greatly depends on the actual images, or even different regions require different window widths. Consequently no optimal window width is guaranteed and choosing one must be done manually, rendering this method less effective.

The main problem of image-fusion based PVE correction algorithms is the assumption that original PET and CT features are highly correlated. This, however, is unrealistic in whole-body scans [9] for technological differences, as PET image is essentially a functional information while CT is structural. PET intensities depend on radiotracer amount in specific regions, depending on blood

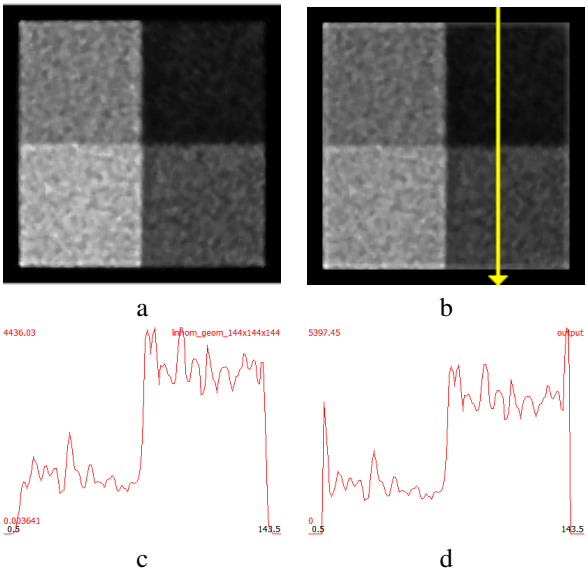


Figure 1: (a) Original PET image (b) Corrected PET image using global scaling and homogeneous CT image (c) Line profile of original PET image (d) Line profile of corrected PET image. Hyperintense edges are present due to uncorrelated images, and neglected local features.

density, metabolism, etc. CT, however, reveals anatomical structures, depending on tissue density. Thus, instead of the correlation of boundaries, a more relaxed connection seems feasible that boundaries being present in both modalities are co-located.

3 Proposed Method

Our goal was to produce a corrected image to enable visual evaluation, which is required in most clinical applications. One purpose of PVC is to enhance region boundaries, hence the resulting image should contain steep edges with increased intensity gradient between regions. Furthermore, a fully 3D algorithm was required as PVE is essentially a 3D effect. Since PET allows quantitative examination of certain physiological functions, corrected images are expected to correctly represent regional uptakes. More specifically, output images should contain regions with total intensities equivalent to original ones. This includes the avoidance of hyperintensive corrected edges earlier seen in Figure 1. Finally, a moderate computational complexity is required to speed up medical evaluation, and the PVC should not take longer than the PET reconstruction. With massively parallel reconstructions available, this time limit around a few minutes for a common 128^3 voxel array.

This paper proposes an algorithm to correct the blurring effect meeting the requirements presented above. The inputs of our PVC algorithm are the PET data and the reg-

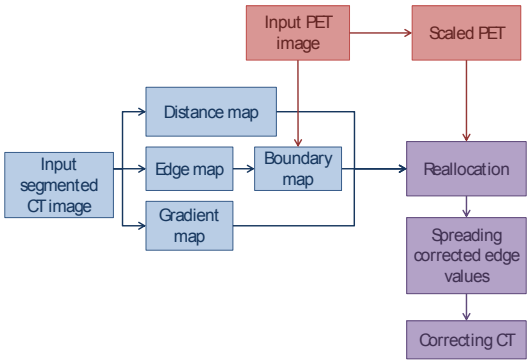


Figure 2: Overview of the proposed PVC algorithm

istered, segmented CT volume. Since CT technology have larger spatial resolution and moderate noise compared to emission tomography, segmentation can be applied to determine CT regions. We applied a custom Kernelized Fuzzy C-Means [11] to the input CT images. Using the region information, the structure of anatomical parts can be taken into consideration. Numerous image segmentation algorithms exist [4, 3, 11] for noisy grayscale images including those for medical applications. In most cases, CT images have larger spatial resolution, thus the PET image is first scaled to the size of the CT image using a simple trilinear interpolation.

The main steps of our proposed algorithm are as follows. To only alter the image where it is necessary, we create an edge map marking edges present in CT, and — using the PET image as well — a boundary map to indicate voxels to be corrected. For the correction we also create a distance map with values marking voxels' distance to the nearest CT segment boundary. Furthermore, a gradient-like map is created for distinguishing boundary voxels with lower or higher intensity compared to nearby regions. Using the gradient-like image we classify voxels in source or spillover regions, i.e. regions with intensity loss or gain.

In the correction step we reallocate intensities with the help of the auxiliary maps created earlier. This is done by extending a 1D concept in 3D. Finally, the CT image is altered based on values calculated during reallocation to compensate offset between PET and CT images. The overview of our PVC algorithm is shown in Figure 2. The steps are discussed in the following subsections.

3.1 Boundary map generation

To fulfil the requirement of only changing the image where necessary, a 3D binary *boundary map* is created, which indicates where the PET volume is worth being corrected. The algorithm starts with an initial binary image marking CT edges. This is saved as an edge map. This boundary map is then filtered by using a standard box-filter and a threshold based on global average PET intensity, leaving voxels where the mean surrounding PET gradient intensity is large enough. This way the image is altered only where PET signal is changing, i.e. where the boundaries are also present in PET image.

The filtered boundary map is then extended by a *morphological dilation* [6] operator. The dilation is required to assure symmetry and constant width at region boundaries. Using directly a box-filter with thresholding, asymmetries may occur depending on PET intensity changes and kernel size. More specifically, the created boundary map may have different widths in segments, possibly leading to erroneous correction. This extended binary map is used as a boundary map later; during the correction only marked voxels are affected.

3.2 Distance map generation

Considering the fact that PVE occurs at the edge of ROIs, the basic principle behind the algorithm is that intensity resembles the original most in the inner part of each region. The general assumption is that CT and PET regions are co-located, thus the PET image can be corrected using CT region structure information. With the help of a CT distance map, PET activities can be labeled based on their proximity to region boundaries.

The algorithm repeats the following procedure for every CT region. The regional distance map is computed by iteratively applying a *morphological erosion* [6] operator. By using an increasing width kernel, the region shrinks in each iteration while preserving the shape. The distance map is completed by assigning to each voxel the last iteration number in which the voxel still belonged to the actual (shrunk) region. This method is illustrated in Figure 3. Clearly, the iteration number determines how far the voxel lies from the nearest region boundary. This way finding the inner parts of each segment is simple, voxels with the highest label number are the innermost ones. It is only necessary to iterate until the region is empty, or — for practical reasons as PVE has limited range — until an iteration limit has been reached. Merging these maps results in the final distance map.

3.3 PET Correction

In the distance map, the larger the value of a voxel in the distance map, the farther it is from the boundary, thus the less affected by PVE. One simple approach would be then, to set every boundary voxel to the innermost average PET intensity. This method sets boundary voxels to the inner region intensity, regardless of original PET activity, leading to incorrect values.

Recall that a basic property of PVE is that higher activity regions cause spillovers to neighboring lower activity regions. In order to maintain average intensity, the algorithm must not change the total intensity, only reallocation

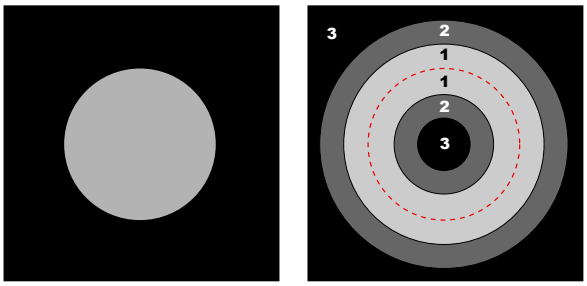


Figure 3: Illustration of the distance mapping. Left: Segmented CT image, a circle in a square. Right: Resulting distance map with three iterations. Darker colors mean higher labels – thus higher distance to boundary (dashed red).

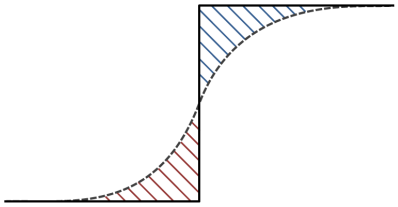


Figure 4: Illustration of the step in *à trous* creating the gradient-like map. At a CT segment boundary the filtered (dashed line) signal is subtracted from the original. The result is positive where the intensity was higher (blue), and negative where the intensity was lower (red). The former is considered as the source region, while the latter is the spillover region.

should occur. The total PET activity in the boundary region is distributed over a larger volume, reaching neighboring CT segments. The basic idea is then that to regain the original activity, these spillovers must be removed and their additional (residual) intensity must be added to their corresponding origin. To do so, spillover regions must be distinguished first.

3.3.1 Identifying Spillover Regions

One step in the 'à trous' [10] wavelet algorithm gives the high-frequency components of the CT image. Due to the method itself, the result is negative at areas with lower activity than its surroundings, exactly where spillover effect occurs. Consequently, classifying based on the sign of CT high frequency components perfectly identifies spillover areas. The concept is depicted in Figure 4. Applying the above method to the segmented CT image (containing segment labels) non-zero values are ensured as segment labels are discrete integers, and the boundary map was created in a way that it corresponds to the low-pass filter of the algorithm, ensuring that positions marked by the boundary map have positive absolute value in this gradient-like map. Note that segment labels must be in order according to average PET intensity for this method to work, but that can be easily achieved during the segmentation process or by a simple initial reordering.

3.3.2 Reallocation

Having identified spillover areas, the next step is to reallocate intensities. In 1D it is a fairly simple task. At each boundary area, spillover residue values must be added to the other – source – side. Since spillover values are determined by the whole boundary region, aggregation must include every value between the two innermost-flagged position. The sum of residues and the sum of values in the higher-intensity part gives the sum of original PET activity in the source (higher-intensity) region. Residue values are calculated by subtracting the innermost value from values of the spillover region. Dividing the sum by the number of positions in the source area results in the corrected — original — activity. This is the corrected value of each position in the source boundary area. On the other side, every boundary position in spillover regions gets the value of the innermost one. This way the residual spillover values are reallocated to their source, and both segment is corrected to their original intensity. This reallocation is depicted in Figure 5.

Observe that distance map assignments are symmetrical to the innermost position. When neighboring positions are grouped by their distance labels it may occur that values from both side in the source region are summed, hence distorting the result, especially when the region acts as source (has higher intensity) at the current edge, but gains spillover values at the other side (having lower intensity). This is the case in Figure 5. For this reason an increasing size window is used. The iteration stops when the innermost label does not change within this square window. It means then that the labels are not increasing, the other side of the segment is in the window.

In higher dimensions (2D and with the same concept in 3D), it is hard to find corresponding spillover areas and sources, due to the interference of neighboring source pixels (voxels). So instead an approximation is used. For every pixel, only the neighboring average original activity is calculated. The idea is that in a square (cube) area (volume) the sum of spillover residue values is the same as the source pixels' corresponding (missing) spillover value sum. Source pixels on the edge of this window cause spillover on pixels outside, though this effect can be neglected for the following reasons.

Assume now that the current pixel to be corrected lies on a long straight axial boundary line (plane), where the two segments are homogeneous, and let the neighboring area be symmetrical to the pixel and the boundary line. This is illustrated in Figure 6. Here every source pixel can be paired with a spillover pixel according to the window's symmetry on the boundary line. The number of spillover

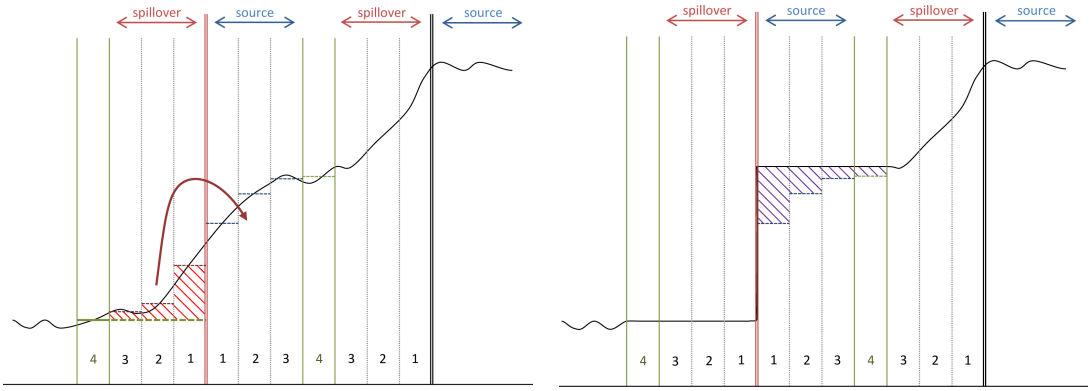


Figure 5: Illustration of the 1D correction algorithm in a general case (left). Positions are labeled by distance map assignments (numbers at the bottom), the innermost is marked by green. The correction transfers spillover residue values (red striped area) to the source. CT segment boundaries are displayed as double lines, and the current boundary is red. Gained values in the result (right) are colored purple.

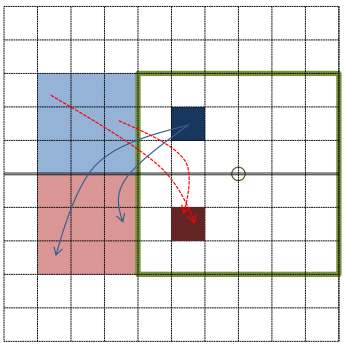


Figure 6: Illustration of how pairing may be done on a symmetrical axial configuration (the window is green). Sources are blue, and spillover pixels are red. The pair has darker color.

pixels related to the source pixel of the pair (meaning the pixels affected by spillover) is the same as vice-versa; the amount of additional intensity of the spillover pixel can be paired with the loss of intensity related to the source pixel of the chosen pair. If the window is not too large, the assumption is not far from reality, and the summing method is correct. Of course the aggregation window must contain the innermost values just as in the 1D case, and interference with the other side of the segment must be avoided.

Another non-trivial task is to calculate the residual spillover value for each pixel. It is much simpler if instead of summing, we use the mean intensity for each distance label. This way the problem is reduced to the 1D case. The correctness can be seen with the assumption that PET activity does not change abruptly within CT segments, so that pixels in the window can be replaced with the mean of their distance label. In addition, this method resolves the problem with non-axial configuration, where different amount of source and spillover pixels may be found. With the averaging, the number of pixels each side are eliminated, and this way they are not affecting the result.

For the same reasons as before, it's hard to determine which pixels should be set to the calculated mean activity. In 1D it was trivial that every position in the source area is set to the calculated intensity, as no interference could occur. In this case, however, values are calculated for a square window around a point on the boundary line (surface in 3D). One could think that applying the same method for inner pixels works the same way, but the position of the window is fixed by the boundary line as values between the two innermost regions are essential for the algorithm. This essentially disallows using the same method. Therefore the idea is that the above method is used only for pixels on the very edge of CT segments, and the values of inner pixels are an interpolation of edge values. Interpolation is done by a distance weighted average, avoiding interference between spillover and source areas by counting them separately. This way, spillover origins are determined evenly without loss of correctness as PET activity is assumed to change non-abruptly in regions.

3.4 Multiple Boundaries

So far only boundaries between two segments were discussed. Handling multiple boundaries within the aggregation window requires some alteration of the algorithm. First of all, multiple spillover areas are affected by one source, possibly in several segments. In order to correct the source value, aggregation should involve all of these spillover areas. Another difference is that the same segment may have both spillover and source areas, depending on its relative intensity to surrounding other segments. Therefore grouping by distance labels must be done separately for spillover and source pixels not to interfere with each other. To ensure that no unrelated boundary inter-

feres with calculations, the iterative window size method must be stopped when the innermost label does not change for segments involved, determining innermost labels separately. Different window sizes for segments does not pose a problem, since the correction is reduced to the 1D case because of the label-based grouping.

In this multiple boundary case, multiple sources jointly determine spillover values nearby. This means that the surrounding residual values must be distributed in accordance with their sources' contribution. More specifically, as the amount of intensity transferred from one higher-intensity to a lower-intensity segment depends on the intensity difference, the total residual value must be split among contributing sources with their average intensity taken into account.

3.5 Fitting CT to PET

As pointed out in [9] the anatomical region may exceed the relevant functional one, as for example tumours may only be metabolically active in a smaller region. This causes underestimation in the PVE correction as more CT positions (in the reduced 1D case) are assumed for the uptake, resulting in a larger denominator explained in 3.3.2.

For this reason, if the process results in a much smaller corrected value than the innermost average, the number of CT positions should be corrected accordingly. More specifically (in the 3D case), CT voxels around the corrected egdes should be reassigned to surrounding CT values (segments) if necessary. Hence the algorithm creates a CT correction map, assigning to each corrected edge voxel the amount of nearby CT voxels (of the same segment as the corrected edge) to be reassigned. Also, the corrected PET values are calculated with the corrected number of CT positions (voxels in 3D).

As a post-processing after the PVE correction, the algorithm reassigns CT voxels – and hence corrected PET values – based on the map created during the correction process. Each voxel on the edges mark the number of CT voxels to be changed nearby. This means that voxels within that radius should be changed to the nearest CT segment, different than the current one. Since the number of CT voxels to be corrected may vary even in small regions, the reassignment of each CT voxel depends on nearby CT correction map values. The altered PET value is an average of the surrounding corrected voxels which is in accordance with the earlier assumptions that PET activity does not change abruptly.

This method can be further extended to compensate even too large corrected values. If the CT and PET images are misaligned, corrected intensities may exceed the innermost values. In small regions this is plausible since they are more affected by PVE, although in larger areas this may indicate an erroneous correction. Therefore the above method is altered that the CT correction map may have negative values, meaning that the current CT segment should be extended there. The corrected PET values are calculated with the appropriate number of CT positions so that the output resembles the innermost values the most. During the CT correction, negative values affect the reassignment decision by reducing the probability of the alteration of voxels in the same segment, and causing other segments to grow less likely.

4 Results

The presented system has been tested on a system with 4 GB RAM and a nVidia GTX 480. The algorithm is inherently parallelisable and is implemented as a sequence of custom filters. To assess the quality of the output we used measurements of a Derenzo phantom and a real mouse.

Figure 9 depicts the Derenzo result. Boundary enhancements are clearly visible, especially at the edge of small tubes. Spillover effects are greatly eliminated except the inner parts of the tube groups, where the innermost average intensity is dominated by spillover values. Some CT segmentation artifacts appear, but these can be reduced with more sophisticated segmentation methods. Results are obtained with a 14 voxel width boundary parameter setting. Increasing the parameter did not cause any further improvement, smaller settings resulted in less sharp edges at the mentioned group centers. As seen in Figure 7, the algorithm results in steep corrected edges without significant overcorrection, in accordance with average original PET activity.

Figure 10 depicts the mouse result. Artifacts appear in soft tissues as CT segments do not cover these slight PET activities. This can be eliminated with proper segmentation. At the spinal area, spillover values are eliminated and reallocated to the source (the spine), and the result is a sharp image. However, near the neck and the head, false voxel classifications are present due to nearby segment boundaries. Since classification is done solely by CT gradients, these edges cause voxels to be classified as source, and due to significant spillover values present nearby, the algorithm amplifies their original value. Increasing the boundary width parameter did not solve this as edges are closer than the parameter already.

Execution time is proportional to the number of voxels on edges (and thus the image size, obviously) and the correction width. This is why the computation time for the mouse took longer than expected (based on the image dimensions). The algorithm produces an output image with steep edges and no significant overcorrection while it preserves local features as opposed to methods earlier mentioned 2.1. Furthermore it requires no *a priori* information on the scanner, nor the tissue being examined. It does not assume any correlation between PET and CT values, only co-located edges are required. A disadvantage of the method is that it may introduce segmentation artifacts. Also, a desirable correction width should be chosen based on the extent of PVE. Classification based on CT gradients may result in erroneous correction as voxels may be

misclassified due to their proximity to an unrelated, lower-intensity CT edge. This could be compensated with a classification taking PET activity into account.

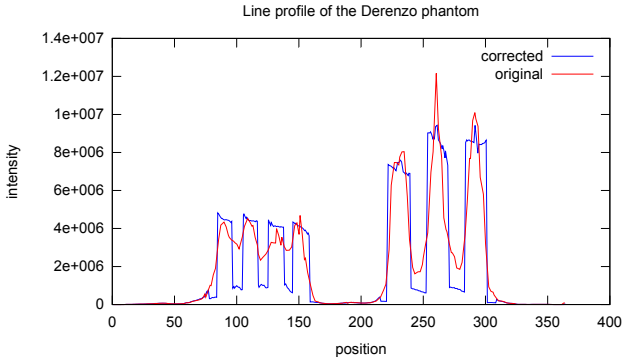


Figure 7: Line profile of the Derenzo phantom.

5 Conclusion

This paper proposed a Partial Volume Effect correction algorithm for PET/CT using segmented CT images. We filter out irrelevant CT features by applying the correction to regions with present radioactivity, and only boundaries are affected. Based on the assumption that PET activity does not change abruptly, and that inner tissue areas represent correct activity, we reallocate PET intensities to restore the original distribution. This is done by extending a 1D method to 3D while original average activity is preserved. The algorithm runs without user interaction and is implemented in a massively parallel framework, harnessing the computational benefits of the GPU.

Acknowledgement

This work has been supported by OTKA K-719922 and the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002).

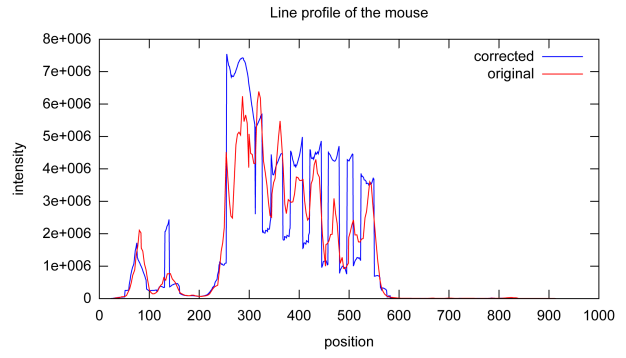


Figure 8: Line profile of the mouse measurement.

References

[1] N. Boussion, M. Hatt, F. Lamare, Y. Bizais, A. Turzo, C. Cheze-Le Rest, and D. Visvikis. A multiresolution image based approach for correction of partial volume effects in emission tomography. In *Physics In Medicine And Biology*, pages 1857–1876. Institue Of Physics Publishing, 2006.

[2] Francisca P. Figueiras, Xavier Jimenez, Deborah Pareto, and Juan D. Gispert. Partial volume correction using an energy multiresolution analysis. In *IEEE Nuclear Science Symposium Conference Record*, pages 2724–2727, 2009.

[3] Zoltan Kato. *Markovian Image Models and their Application in Image Segmentation*. PhD thesis, University of Szeged, Hungary, 2007.

[4] Prabhjot Kaur, Dr. I. M. S. Lamba, and Dr. Anjana Gosain. A robust method for image segmentation of noisy digital images. In *2011 IEEE International Conference on Fuzzy Systems*. IEEE, 2011.

[5] John M. Ollinger and Jeffrey A. Fessler. Positron-emission tomography. *IEEE Signal Processing Magazine*, pages 43–55, 1997.

[6] Gerhard X. Ritter and Joseph N. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, second edition, 2001.

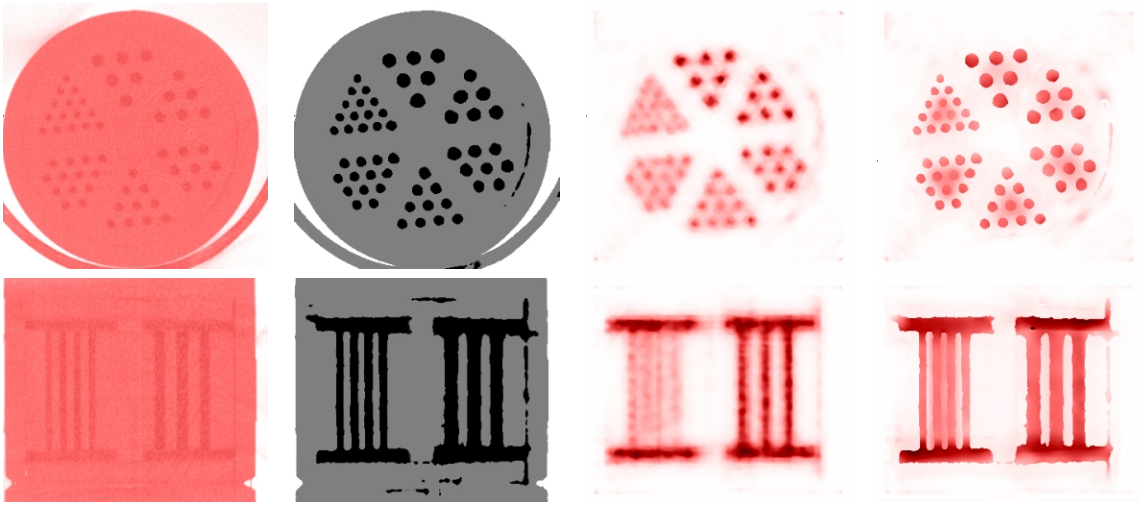
[7] Olivier G. Roussel, Yilong Ma, and Alan C. Evans. Correction for partial volume effects in pet: Principle and validation. *Journal of Nuclear Medicine*, 39:904–911, 1998.

[8] Miho Shidahara, Charalampos Tsoumpas, Alexander Hammers, Nicolas Boussion, Dimitris Visvikis, Tetsuya Suhara, Iwao Kanno, and Federico E. Turkheimer. Functional and structural synergy for resolution recovery and partial volume correction in brain pet. *Elsevier NeuroImage*, 2009.

[9] Marine Soret, Stephen L. Bacharach, and Irène Buvat. Partial-volume effect in pet tumor imaging. *Journal of Nuclear Medicine*, 2007.

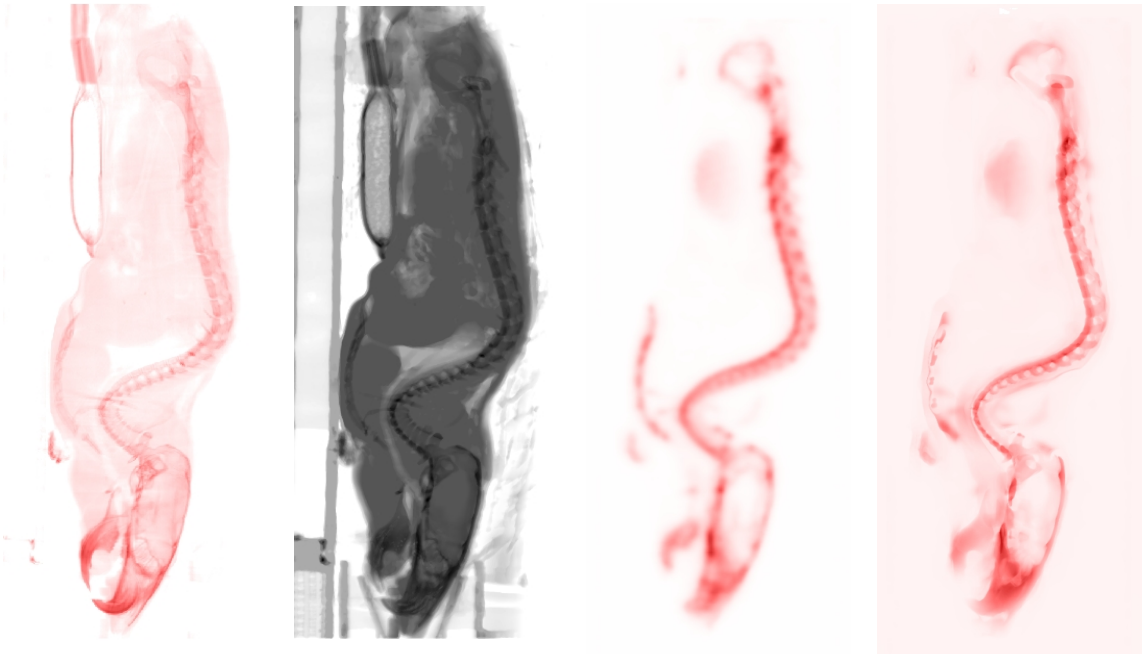
[10] Jean-Luc Starck, Fionn Murtagh, and A. Bijaoui. *Image Processing and Data Analysis, The Multiscale Approach*. Cambridge University Press, 1998.

[11] Dao-Qiang Zhang and Song-Can Chen. A novel kernelized fuzzy c-means algorithm with application in medical image segmentation. *Elsevier Artificial Intelligence in Medicine*, pages 37–50, 2004.



CT: $333 \times 333 \times 281$ Segmented CT PET: $173 \times 173 \times 146$ Corrected PET

Figure 9: Slices of the Derenzo measurement and steps of correction. Execution time is around 3 minutes. Boundary width is 14 voxels.



CT: $435 \times 422 \times 1056$ Segmented CT PET: $136 \times 132 \times 330$ Corrected PET

Figure 10: Blended slices from the mouse measurement and the steps of correction. Execution time is around 40 minutes. Boundary width is 19 voxels.

Flow-based Segmentation of Seismic Data

Kari Ringdal*
Supervised by: Daniel Patel†

Institute of Informatics
University of Bergen
Bergen / Norway

Abstract

This paper presents an image processing method for identifying separate layers in seismic 3D reflection volumes. This is done by applying techniques from flow visualization and using GPU acceleration. Sound waves are used for exploring the earth beneath the surface. The resulting seismic data gives us a representation of sedimentary rocks. Analysing sedimentary rocks and their layering can reveal major historical events, such as earth crust movement, climate change or evolutionary change. Sedimentary rocks are also important as a source of natural resources like coal, fossil fuels, drinking water and ores. The first step in analysing seismic reflection data is to find the borders between sedimentary units that originated at different times. This paper presents a technique for detecting separating borders in 3D seismic data. Layers belonging to different units can not always be detected on a local scale. Our presented technique avoids the shortcoming of existing methods working on a local scale by addressing the data globally. We utilize methods from the fields of flow visualization and image processing. We describe a border detection algorithm, as well as a general programming pipeline for preprocessing the data on the graphics card. Our GPU processing supports fast filtering of the data and a real-time update of the viewed volume slice when parameters are adjusted.

Keywords: Seismic Data, Structure Extraction, GPU-accelerated Image Processing

1 Introduction

Stratigraphy is the study of rock layers deposited in the earth. A stratum (plural: strata) can be defined as a homogeneous bed of sedimentary rock. Stratigraphy has been a geological discipline ever since the 17th century, and was pioneered by the Danish geologist Nicholas Steno (1638-1686). He reasoned that rock strata were formed when particles in a fluid, such as water, fell to the bottom [10]. The particles would settle evenly in horizontal layers on a lake or ocean floor. Through all of Earth's history, layers of sedimentary rock have been formed as wind, water or ice has deposited organic and mineral matter into a body of water. The matter has sunk to the bottom and consolidated into rock by pressure and heat. A break in the continuous deposit results in an *unconformity*, in other words, the surface where successive layers of sediments from different times meet. An unconformity represents a gap in the geological record. It usually occurs as a response to change in the water or sea level. Lower water levels expose strata to erosion, and a rise in the water level may cause new horizontal layers of deposition to resurge on top of the older truncated layers. The geologists analyse the pattern around an unconformity to decode the missing time it represents. Above and below an unconformity there are two types of terminating patterns and one non-terminating pat-

Figure 1: Stratified sediments. The sedimentary facies are separated by unconformities.

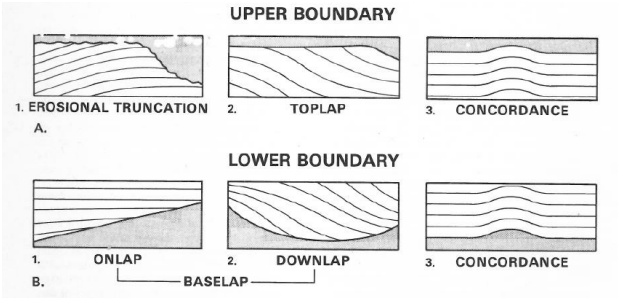


Figure 2: Strata relates to an unconformity in different ways. The top three images show patterns that occur below an unconformity, and the bottom three images show patterns that occur above an unconformity.

*kari.ringdal@student.uib.no
†danielpatel.no@gmail.com

tern. Truncation and toplap terminates at the unconformity above. Truncation is mainly a result of erosion and toplap is a result of non-deposition. Strata above an unconformity may terminate in the pattern of onlap or downlap. Onlap happens when the horizontal strata terminates against a base with greater inclination, and downlap is seen where younger non-horizontal layers terminates against the unconformity below. Concordance can occur both above and below an unconformity and is where the strata layers are parallel to the unconformity. An illustration of these concepts can be seen in Figure 2.

An unconformity can be traced into its *correlative conformity*. In contrast to an unconformity, there is no evidence of erosion or non-deposition along the conformity.

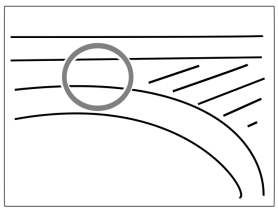


Figure 3: Sediments of different facies can be indistinguishable in local areas such as inside the circle.

A seismic sequence - also called a sedimentary unit or facies, is delimited by unconformities and their correlative conformities. The fact that sediments belonging to different seismic sequences can be indistinguishable in greater parts of the picture, as illustrated in Figure 3, calls for global analysing tools.

For more information on unconformities, sedimentary sequences and other stratigraphic concepts, the reader is referred to Nichols book on sedimentology and stratigraphy [13] and Catuneaus book on sequence stratigraphy [4].

Many techniques to highlight interesting attributes in seismic data have been developed [6]. Taner [17] gives a useful definition of seismic attributes: "Seismic Attributes are all the information obtained from seismic data, either by direct measurement or by logical or experience based reasoning". *Dip* and *azimuth* are attributes that describe the dominating orientation of the strata locally. Dip gives the vertical angle and azimuth the lateral angle.

In our work, we consider the dip/azimuth vectors to constitute a flow field representation of the data set where the flow "moves" along the sediment layers. We seed particles from neighbouring points in this flow, and consider the distances between the end points of the generated trajectories. A great distance gives a high probability that the seed point is a surface point. Mapped surface points are then linked to constitute unconformity surfaces. Figure 4 gives an overview of our processing pipeline.

2 Related Work

Interpreting seismic data is a time consuming task, and extensive work has been done to automate this. This section will focus on previous approaches on finding unconformities, and also look into methods within the fields of image processing and flow visualization that relates to the new technique presented in this paper. Orientation field extraction from image processing relates to vector field extraction of seismic data. Image processing also deals with edge linking methods, which relates to the segmentation process of our method. The field of flow visualization use methods relevant to the mapping of surface probability.

2.1 Seismic methods

One method for detecting sequence boundaries, or unconformities, is the method of Randen at al. [15]. This method calculates the volumetric estimates of dip and azimuth by applying a multi-dimensional derivative of a Gaussian filter followed by directional smoothing. Starting at a sample in the extracted orientation field, a curve is generated in the direction of the local angle (see Figure 5).

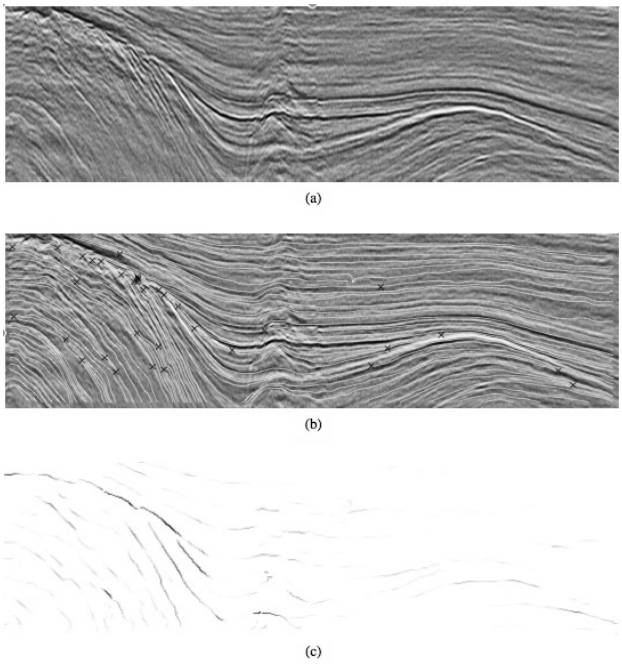


Figure 5: a) Seismic cross section. b) Flow lines and terminations (marked with X) extracted from the cross section. c) Mapping of stratigraphic surface probability from the cross section.

The curves form flow lines along the orientation field. Intersection points of flow lines are detected (marked with X). These points are likely to be on an unconformity. Brown and Clapp attempted a different approach that locally compares the data to a template that represents a neighbourhood around an unconformity [2]. Another method to find lateral discontinuities (e.g. lateral unconformities and faults) in seismic data is that of Bahorich and Farmer called the coherence cube [1]. Coherency coefficients are calculated from a 3D reflection volume and displayed as a new volume. Coherence is a measure of the lateral change of the seismic waveform along structural dip. Since the coherence cube first appeared in 1995 it has been

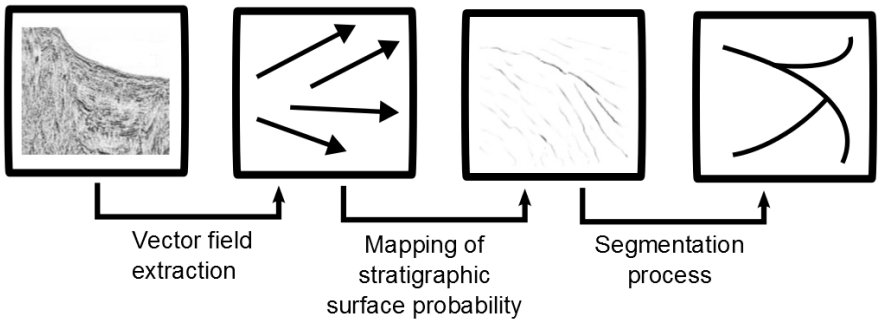


Figure 4: Processing pipeline. A flow field with vectors parallel to the sediment layers is extracted from the data. Stratigraphic surface probability is mapped by the use of a seeding algorithm and user defined parameters. Edge linking can be a way of completing a segmentation.

improved several times. Chopra gives an overview of the development of this method [5]. Unconformities are often seen as discontinuities in the data, but not always. It can happen that there are no obvious signs of erosion and the layers on either side of an unconformity are parallel. This type of unconformity, sometimes called paraconformity or conformity, would not be detected by the coherence cube or any of the above methods.

A recent paper by Hoek et al. [18] describes a new method for finding unconformities. Gaussian derivative filters are used to estimate the dip/azimuth field. The orientation field is then analysed by utilizing a method from the field of image processing. The structure tensor field is calculated and regularized, and the principal eigenvector of the structure tensor is extracted. From this, the dip field is studied to see whether the vectors diverge, converge or are parallel. Hoek et al. recognize the problem that previous methods address seismic data on a local scale, and they attempt to find a more global approach with their unconformity attribute. However, their method measures the conformability of the dip field in a neighbourhood of a predefined size and is therefore still a local method that does not capture events taking place outside its neighbourhood.

Hoek et al.'s unconformity detection method, as well as the method presented in this paper, depends on the reflector dip and azimuth of the seismic data. Much work has been done to extract these attributes accurately. Complex trace analysis, discrete vector dip scan, and gradient structure tensor are commonly used for the task. Marfurt presents a refined method for estimating reflector dip and azimuth in 3D seismic data and gives a good overview of the work previously done in this area [12].

2.2 Image processing methods

In image processing, a repetitive pattern is referred to as a texture, and a linear pattern as an oriented texture. Numerous algorithms are used for enhancing or segmenting textured images - many inspired by human visual perception models. When it comes to processing images digitally for tasks such as edge detection or pattern recognition, there is no algorithm generic enough to be a good choice at all times. It is in other words necessary to choose the right algorithm for the right data and desired achievement.

A finger print, wood grain or a seismic image are examples of oriented textures. These textures have a dominant orientation in every location of the image. They are anisotropic, and each point in the image has a degree of anisotropy that relates to the rate of change in the neighbourhood. This is often represented as the magnitude of the orientation vectors. Extraction of the orientation field of a texture has been well researched in the field of image processing. Extraction algorithms are often based on the gradient from a Gaussian filter [16, 9]. In addition to automatic pattern recognition and some edge detection algorithms, directional smoothing of an image exploits the orientation field. Like in seismic data evaluation, it is essential that the extracted orientation field represents the intrinsic properties of the image.

Non photo-realistic rendering (NPR) concerns with simplifying visual cues of an image to communicate certain aspects more effectively. Kang et al. [8] suggests a new NPR method for 2D images that uses a flow-based filtering framework. An anisotropic kernel that describes the flow of salient image features is employed. They presents two topics that are interesting to our technique, namely the extraction of a vector field from an image, and creating lines from isolated points. They use a bilateral filter (an edge-preserving smoothing filter) for the construction of what they call an edge tangent field (ETF). This is a vector field perpendicular to the image gradients. The gradient map is obtained by a Sobel operator. The vector adapted bilateral filter takes care to preserve salient edge directions, and to preserve sharp corners in the input image. The filter may be iteratively applied to smooth the ETF without changing the gradient magnitude. Kang et al. present this vector field extraction method as a base for extracting image features. A different vector field extraction method is proposed by Ma and Manjunath [11]. They find edge flow vectors by identifying and integrating the direction of change in color, texture and phase discontinuity at each image location. The vector points in the direction of

the closest boundary pixel.

Edge linking is another area from image processing that is relevant to our method. An image of unconformity lines may contain gaps, and with an ultimate goal of segmentation, proper linking of edges is necessary. Fundamental approaches to edge linking concern both local processing where knowledge about edge points in a local region is required, and regional processing where points on the boundary of a region must be known [7]. There are also global processing methods, like the Hough transform. For Kang et al.'s flow-based image abstraction method [8], part of the goal is to end up with an image-guided 2D line drawing. Here the lines are generated by steering a DoG (difference of Gaussian) edge detection filter along the ETF flow and accumulate the information. This way, the quality of lines is enhanced. (see Figure 6).

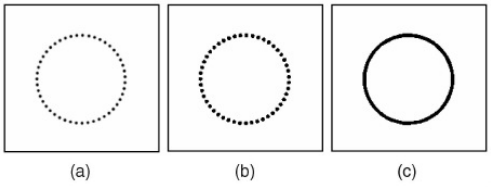


Figure 6: Edge linking. a) Input image. b) Filtered by DoG filter. c) Filtered by Kang et al.'s flow-based DoG filter.

2.3 Flow field topology and extraction methods

Flow visualization is a sub-field of data visualization that develops methods to make flow patterns in fluids visible. Flow features and techniques for topology extraction of steady vector field data will be the focus of this section. A feature is a structure or an object of interest. Shock waves, vortices, boundary layers, recirculation zones, and attachment and separation lines are examples of flow features.

Relating flow to seismic data, features that are most likely to occur in a vector field extracted from seismic data are separation and attachment lines, this because of the onlap, toplap and downlap terminations. Separation and attraction lines are lines on the boundary of a body of a flow where the flow abruptly moves away from or returns to the surface of the flow body. A state of the art report by Post et al. [14] deals with different methods for separation and attachment line extraction. Methods for both open and closed separation are discussed. One approach mentioned is particle seeding and computation of integral curves. A particle is released into the flow field and its path is found by integrating the vector field (that represents the flow field) along a curve. If we look at the vector field extracted from seismic data as a flow field, we have a steady flow. The fact that it is not time-dependent means that the pathline of a seeded particle is everywhere tangent to the vectors of the flow. According to the aim of this paper, feature extraction and its instrumental algorithms are of greater interest than the actual visualization of the data. We will not use pathlines for visualization purposes, but as a tool in addressing the seismic data on a global scale.

3 Implementation Details

Our method for separating the sedimentary units in 3D seismic data follows the processing pipeline shown in Figure 4. The processing steps are separated into three categories:

- Vector field extraction from the seismic data
- Mapping of stratigraphic surface probability
- Segmentation process

This section will take a closer look at each of the steps, but focus on the second step of the pipeline - the mapping of stratigraphic surface probability using concepts from flow. In this step lies the novelty of our method.

3.1 Preprocessing - vector field extraction on the GPU

As described in the Related work section, there already exists efficient methods for estimating the reflector dip and azimuth of seismic data. The extraction of this vector field is an important step of the pipeline, because a regular vector field, that represents the data accurately, is crucial to our technique. The idea is to apply our unconformity extraction algorithm on a dip/azimuth vector field found by already established methods. For testing the algorithm we have created 2D and 3D synthetic data sets and implemented image processing methods for extracting the vector field. The 3D data sets were created either by stacking an image to form a volume or by procedurally creating a volume of vectors pointing in different directions on either side of a delimiting surface. Only the last mentioned type of test set has a variation in z-direction. The test sets have been useful for testing our algorithm and for getting a feel of the vector field extraction calculations which were done in parallel on a graphics card. The implemented programming pipeline is transferable to real seismic data volumes.

The highly parallel structure of modern GPUs is ideal for efficiently processing large blocks of data provided the calculations could be done in parallel. GPUs support programmable shaders that manipulate geometric vertices and fragments and output a color for every pixel on the screen. Instead of output to the screen, the RGBA-vectors can be written to 2D or 3D textures. A texture is in this sense a block of memory located on the GPU where every point of the texture is a four dimensional vector. GPU-accelerated methods are rapidly expanding within the oil and gas industry, and have dramatically increased the computing performance on seismic data.

For our implementation, we have used the OpenGL API and the OpenGL shading language GLSL within the framework of Volumeshop [3]. The programmed pipeline does GPU filtering of 3D data iteratively with a real-time update of the viewed 2D slice when adding, removing or adjusting any of the filters. This allows a fine-tuning of parameters before the entire data volume is processed.

The volume is loaded on the GPU in a 3D texture. GPU-memory is reserved for two more textures of equal size as the volume, and the original texture is copied to one of them. The two textures are used alternately for reading and writing in a ping-pong fashion while the desired number of filters is applied. The different filters are written as shaders in a plugin. Any number of this plugin is added to the Volumeshop interface. They all operate on the GPU by having one plugin for every filter applied to the volume. The user chooses a filter from a pull down menu, and the filter parameters can be adjusted by sliders. It is also possible to choose how many slices of the volume are filtered at each step. This way 3D filtering can be done on a subset of the volume to quickly assess the result. Adjustments of a filter leads to reprocessing of the data from the original volume through all the added filters. Therefore, the original volume is kept as a separate texture on the GPU and not overwritten as the ping-pong textures. Figure 7 illustrates the implementation. All calculations are saved with floating point precision in the RGBA-vectors of the 3D textures. Results can be visualised directly by rendering the RGBA-vectors to the screen, i.e. a flow field is seen as colors that varies according to the dip/azimuth vectors. This gives a good indication of the effect of each applied filter. A flow field represented as colors can be seen in Colour plate Figure 12 b).

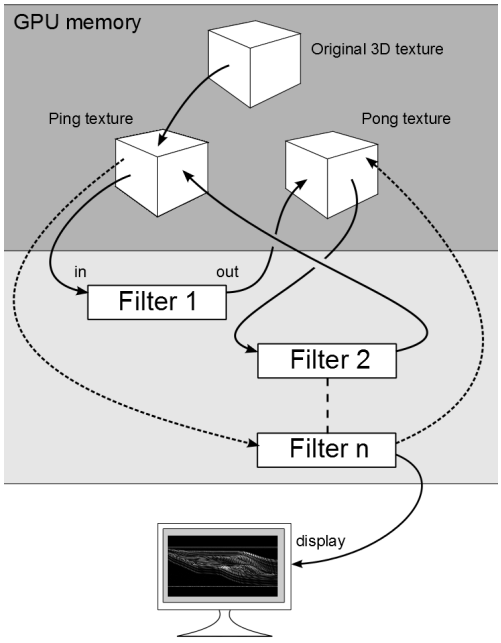


Figure 7: The data is loaded onto the GPU memory in a 3D texture. Data is alternately read and written between two more textures in a ping-pong fashion while filters are applied iteratively. The data is filtered in parallel on the GPU, and a flow field representing the data is extracted. Output from a filter is rendered to the ping or pong textures. Any output slice can also be rendered to the display.

3.2 Mapping of stratigraphic surface probability

Our unconformity detection algorithm is implemented as a shader, and calculations are done in parallel on the GPU. The technique uses particle seeding, as is common in the field of flow visualization, but the paths of the particles are not visualized. We use particle seeding from four neighbouring points to check whether they belong to the same sedimentary unit or not. The algorithm is as follows: For every point on a volume slice, four seed points are chosen. The seed points have coordinates (x,y,z), $(x+d,y,z)$, $(x,y+d,z)$ and $(x,y,z+d)$. d is set so all four points are within a local neighbourhood. The seeding is calculated by sampling the flow from the 3D texture using the Runge Kutta 4th order (RK4) method. All four paths are followed until a user- defined number of steps are taken, or the path reaches coordinates outside of the volume-texture. The distances between the four end points are calculated. If a distance greater than a user-defined threshold is detected, the original seed point with coordinates (x,y,z) is marked as a probable point on an unconformity. Figure 8 is an illustration of the algorithm in 2D. It is expected that a great distance indicate that the particles have moved along differently shaped paths. Because of the parallel nature of the seismic data, two close paths will end up in the same neighbourhood if their seed points are within the same sedimentary unit. We use four seed points to detect borders of any angle. The advantage of this method is that paths of many steps address the data globally and unconformities can be mapped even with parallel horizons on both sides. A premise is that the particles move into a non-parallel area along their paths.

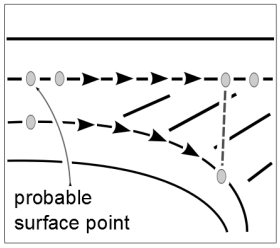


Figure 8: Illustration of the border detection algorithm. The dots represents three seeds that move along paths in a flow field. If the distance between the path-ends exceeds a threshold, a probable surface point is marked.

Since the algorithm is implemented as a shader, it is executed in parallel for every pixel of the rendered region. The rectangular viewing region is set to correspond to the

width and height of the volume, and this region is rendered to the screen or directly to the ping-pong textures. The user can adjust the number of path steps and the distance threshold used by the algorithm. The effect of adjusting these parameters is seen in real-time on one slice of the volume. When the desired parameter values are set, the rest of the volume is processed, slice-by-slice, with the same settings. This way, the seeding algorithm is executed for every vertex in the volume.

When the flow field is sampled from the 3D texture during the path calculations, OpenGL takes care of any necessary interpolation. The textures are set up for trilinear interpolation, and the paths are found by the RK4 integration method. If the step size of this method is set to h, the error per step is on the order of h^5, while the total accumulated error has order h^4. Because of the error accumulation, a small step size is desirable. The step size together with the number of steps affect whether the implemented technique is run locally or globally. Since the RK4 calculations are a bottleneck in our technique, the balancing between the accuracy of the method and the performance speed lies in the choice of these parameters. We are using a step size of 0.5. The number of steps is chosen in the user interface.

Ideally, the mapped points constitute unconformity lines without gaps when the seeding is done for every point on a slice, and unconformity surfaces without gaps when the seeding is done on the entire volume. However, this is rare, when dealing with real seismic data, and some fragmentation and false positives may occur. We have taken some measures to reduce misclassification. For more robustness of the algorithm we perform the seeding in both directions of the flow. We also scale the endpoint distances by the number of path steps to makes sure that a path of few steps will be as sensitive to the given threshold as a path of many steps. To avoid false positives, compared particle paths are discarded if their paths have a great variation in the executed number of steps.

3.3 The segmentation process

The ultimate goal is a fully automated segmentation of seismic data into sedimentary units. However, when running the algorithm on noisy data, some fragmentation of the detected borders may occur. In that case, edge linking is required. This step is not implemented at the current stage, as it is not the main focus of this work. See the end of Section 2.2 for possible methods to achieve edge linking.

4 Results / Application

To test the idea behind the presented technique, 2D and 3D test sets were generated. The first 2D test set of size 256×256 is a vector field that simulates the flow field of two sedimentary units with parallel layers in greater parts of the picture. As expected, the unconformity surface was only found in the area without parallel layers when the algorithm was run locally (see Figure 9a). Each particle path had 20 steps with a step size of 1 which means that every particle travelled within a local neighbourhood. Figure 9b shows the result of increasing the number of steps to 200. Now the unconformity was mapped all the way, also in the area of parallel layers.

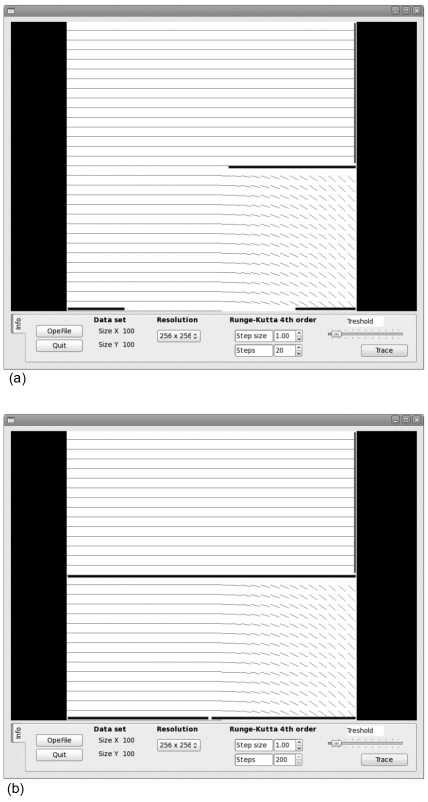


Figure 9: 2D test set of size 256×256 representing the vector field from two sedimentary units. a) The border detection algorithm is run locally - each particle is followed for 20 steps, and only a part of the border is detected. b) The algorithm is run globally - each particle is followed for 200 steps, and the border is detected throughout the data set.

The second test set is a 3D vector field of size $256 \times 256 \times 256$. It also simulates two sedimentary units. Figure 10 a) shows the flow field on a slice of the volume. The two units have vectors with different z-components and the data set varies in depth. The seeding algorithm was run on this flow field with 100 path steps and a step size of 0.5, and the unconformity surface was found. Figure 10b shows the detected surface displayed in VoulmeShop. On this simple test set, the algorithm has generated a complete surface, and edge linking is not required.

For test set number 3 we used an image of a seismic data slice from Randen et al.'s article [15], and stacked copies of this image to create a 3D volume. The resulting volume is of size $256 \times 256 \times 256$. This data set does not vary in depth as a seismic volume, but it gives us a

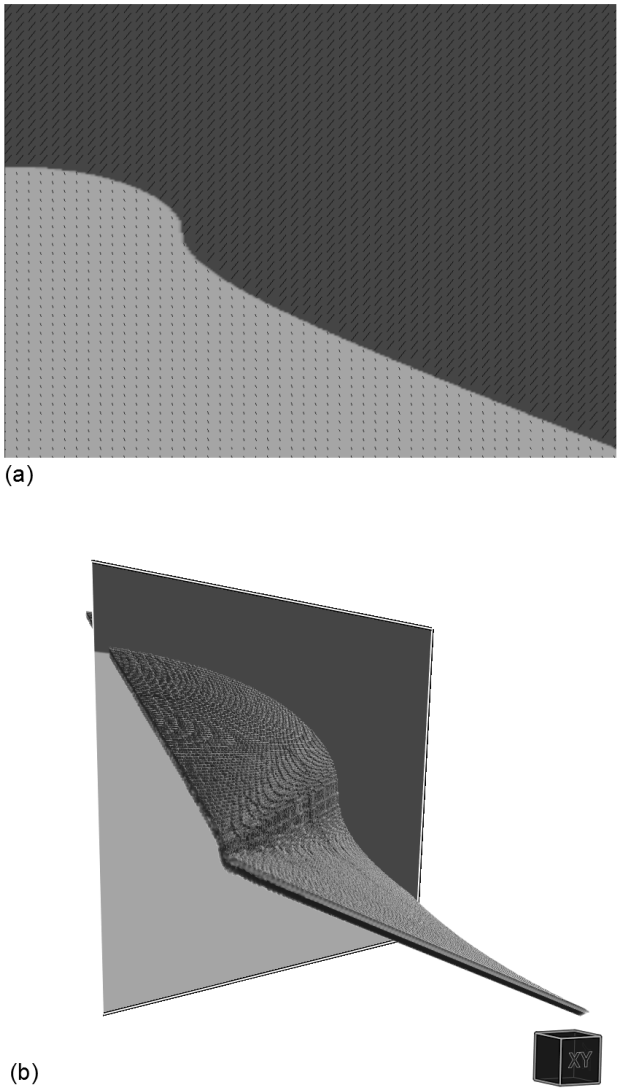


Figure 10: Test set 2 is a 3D vector field. a) A slice of the volume with lines following the flow field. b) A volumetric representation of the surface extracted by the border detection algorithm.

way to test our vector field extraction filters and 3D filtering in real-time. Four filters were applied to the test set. Gradients were calculated by central difference and rotated 90 degrees clockwise around the z-axis. Flow vectors with a negative x-component were turned 180 degrees. Then a Gaussian blur filter was used before a median filter smoothed the flow field even more. At last we applied the filter that contains the border detection algorithm. The result is seen in Figure 11, and shows that some borders are detected also in places where the layers appear parallel.

The last test set is again generated from stacking copies of an image. We used an image that represents a seismic image picturing many sedimentary units (Colour plate Figure 12a). The data set is of size $512 \times 512 \times 64$. First the flow vectors were found by central difference as with test set number 3. Then the flow field was smoothed by Kang et al.'s bilateral smoothing filter [8] reviewed in Section 2.2. Also, a 3×3 mean filter was used for more smoothing. All vectors were normalized. Colour plate Figure 12b is a rendering of the RGBA-vectors constituting the flow field. Colour plate Figure 12c and d shows the output of the border detection algorithm with two different threshold settings. The step size is 0.5 and the number of steps is 1000 in both cases. A path may evolve for less than 1000 steps if the path reaches the edge of the volume. The distance threshold in Colour plate Figure 12c is set so that the algorithm maps any seeding point where the seeded particles diverged for more than 16 units at the path ends. In Colour plate Figure 12d the threshold is set to 4 units. Clearly, a smaller threshold maps more points as a border point. The effect of changing the threshold value can be seen in real-time when the algorithm is run on one slice of the volume due to our GPU implementation. Therefore, the user can find a satisfactory threshold value before the entire volume is processed.

Testing was done on a machine with an *NVIDIA GeForce GXT* 295 graphics card with a global memory of 896 MB and an *Intel*^{R} *Core*^{TM} 2 *Duo* processor with 2GB ram. Running times for the last and biggest test set of size $512 \times 512 \times 64$ was as follows: It took 2.4 seconds to filter the whole set by the three flow field extraction filters and running the seeding algorithm with 1000 path steps on one slice. When the seeding algorithm was run on all 64 slices, the same process took 44.5 seconds. The implementation was written in C++ and OpenGL within the framework of Volumeshop. All calculations are done on the GPU.

5 Conclusions and future work

The paper has demonstrated an automated method for highlighting unconformities in seismic data. An implementation of the technique, according to given implementation details, has shown a promising outcome in four different tests. Although the test sets were quite small, manually generated and more regularized than most seismic data sets, the presented implementation shows a possible way of detecting seismic unconformities on a global scale. We have also presented a programming pipeline for processing seismic data with all calculations done on the GPU. Two OpenGL 3D textures are used in a ping-pong fashion for reading and writing while filtering the data volume. The volume is updated in real-time when applying or adapting any filters.

For testing with actual seismic data sets, a few improvements to the program are needed. Seismic data sets are often very large in size, and it would not be possible to load such volumes in entirety onto the GPU. A method for processing large data sets in sequence, such as stream-processing, is necessary. We would also like to refine the detection algorithm itself to better handle the possible features of a flow extracted from seismic data.

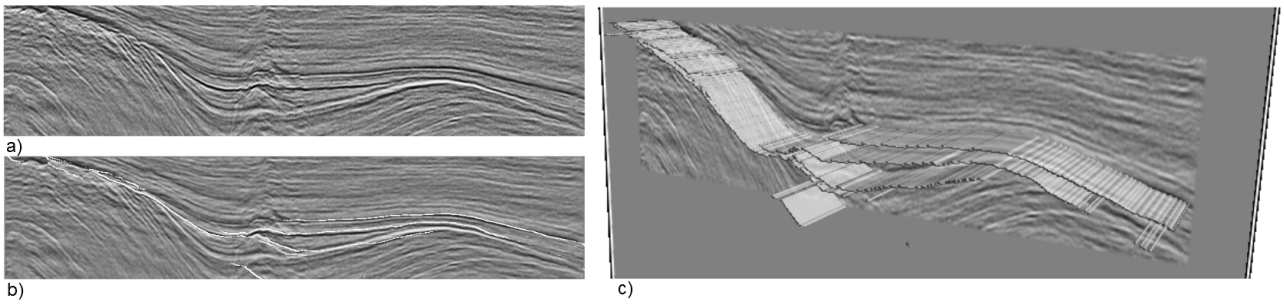


Figure 11: a) A seismic cross section. This image is copied and stacked to constitute a 3D test set. b) One slice of the output volume after our border detection algorithm is employed. The detected border is displayed as a white line. c) The detected border is seen as a surface when displayed in VolumeShop. A slice of the volume is added for reference.

Acknowledgement

Thanks to Danel Patel, who supervised this project, for all his help and inspiring discussions.

References

[1] M. Bahorich and S. Farmer. The Coherence Cube. *The Leading Edge*, 14(October):1053–1058, 1995.

[2] M. Brown and R. G. Clapp. Seismic pattern recognition via predictive signal/noise separation. *Stanford Exploration Project, Report 102*, pages 177–187, 1999.

[3] S. Bruckner and M. E. Groller. VolumeShop: an interactive system for direct volume illustration. *IEEE Visualization '05* , pages 671–678, 2005.

[4] O. Catuneanu. *Principles of sequence stratigraphy*. Elsevier, 2006.

[5] S. Chopra. Coherence Cube and Beyond. *First Break*, 20(1):27–33, 2002.

[6] S. Chopra and K. J. Marfurt. Seismic attributes - A historical prespective. *Geophysics*, 70(5):3–28, 2005.

[7] R. C. Gonzalez and R. E. Woods. *Digital image processing*. Prentice Hall, 2008.

[8] H. Kang, S. Lee, and C. K. Chui. Flow-based image abstraction. *IEEE transactions on visualization and computer graphics*, 15(1):62–76, 2009.

[9] M. Kass and A. P. Witkin. Analyzing oriented patterns. *In Proc. Ninth IJCAI* , pages 944–952, 1985.

[10] H. Kermit. *Niels Stensen, 1638-1686: the scientist who was beatified*. Gracewing Publishing, 2003.

[11] W. Y. Ma and B. S. Manjunath. EdgeFlow: a technique for boundary detection and image segmentation. *IEEE transactions on image processing*, 9(8):1375–88, January 2000.

[12] K. J. Marfurt. Robust Estimates of 3D Reflector Dip and Azimuth. *Geophysics*, 71(4):P29–P40, 2006.

[13] G. Nichols. *Sedimentology and stratigraphy*. John Wiley and Sons, 2009.

[14] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The State of the Art in Flow Visualisation: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[15] T. Randen, B. Reymond, H. I. Sjulstad, and L. Sonneland. New seismic attributes for automated stratigraphic facies boundary detection. In *SEG Expanded Abstracts, Soc. Exploration Geophys, INT 3.2*, volume 17, pages 628–631, 1998.

[16] A. R. Rao and B. G. Schunck. Computing oriented texture fields. *CVGIP: Graphical Models and Image Processing* , 53:157–185, 1991.

[17] M. T. Taner. Seismic attributes. *Canadian Society of Exploration Geophysicists Recorder*, 26(9):48–56, 2001.

[18] T. (Shell Malaysia E&P) van Hoek, S. (Shell International E&P) Gesbert, and J. (Shell International E&P) Pickens. Geometric attributes for seismic stratigraphy interpretation. *SEG, The Leading Edge*, 29(9):1056–1065, 2010.

Stochastic Particle-Based Volume Rendering

Philip Voglreiter*
Supervised by: Bernhard Kainz†

Institute for Computer Graphics and Vision
Graz University of Technology
Graz / Austria

Abstract

In this paper we propose a particle-based volume rendering approach for unstructured three-dimensional tetrahedral polygon meshes. We stochastically generate millions of particles per second that are projected on the screen in real-time. In contrast to previous rendering techniques of tetrahedral volume meshes, our method does not need a prior depth sorting of geometry. Instead, the rendered image is generated by choosing particles closest to the camera. Furthermore, we use spatial superimposing. Each pixel is constructed from multiple subpixels. This approach not only increases projection accuracy, but allows also a combination of subpixels into one superpixel that creates the well-known translucency effect of volume rendering. We show that our method is fast enough for the visualization of unstructured three-dimensional grids with hard real-time constraints and that it scales well for a high number of particles.

Keywords: volume rendering, GPGPU, particle-based, object space

1 Introduction

Volume rendering is used in many disciplines. Visualization of medical data or simulated data arising from finite element methods are just a few examples. The data to be rendered can be represented as regular or irregular structure. Regularly structured data originates mostly from medical imaging devices (MRI, CT, etc.). Direct visualization of these volumetric datasets is well researched and various methods exist. A good overview over standard methods is given by Hadwiger *et al.* [8].

Irregular datasets – or unstructured grids –, are mainly used for simulations, for example for finite element analysis [3], which normally uses an input of irregular shape and which requires connectivity information of the grid's nodes. Rendering such grids is an ongoing field of research. Early approaches use standard geometric algorithms such as plane sweep techniques. Other algorithms directly exploit the grid structure. Using tetrahedral grids is the most prominent method. The grids can either be projected directly, or they can be rendered in a preprocessed state to speed up the rendering process [14].

Basically, all methods for volume rendering can be divided into two main areas. They either are *image-based* or *object-based*. Image-based methods, like ray casting, generally scale with image resolution. Their performance highly depends on the amount of pixels to be displayed. In contrast, object-based approaches like point splatting are less dependent on image resolution. The complexity of rendering is strongly tied to volume complexity. Particle-based volume rendering (PBVR) belongs to the object space approaches. In contrast to many other methods in this category, PBVR does not require depth sorting of any kind. Instead, we treat projected particles in a way that is similar to z-buffering.

Modern applications demand fast visualization techniques. Real-time generation of images with an acceptable frame rate is essential for visualization techniques such as Augmented Reality [5] or other applications with hard real-time contstraints. Often, several tasks need to be performed in parallel. Especially medical applications need to provide a wide field of techniques concurrently to the visualization of data. Recorded images often need to be segmented. Also, simulations need to be performed simultaneously to visualization.

Modern GPUs give an option for fast visualization methods and allow solving a vast amount of parallel problems in real-time. In this paper, we introduce a novel way of stochastic PBVR on modern GPUs. In contrast to the highly sophisticated particle generation methods (Metropolis [12]) used by former approaches, we introduce a method of particle generation with little computational effort. Our proposed method also allows online control of the number of generated particles. This is crucial for applications with hard real-time constraints and allows to alter visual effects such as density during runtime. Because the number of particles also influences the computational effort and memory consumption, our proposed online control can also be used to steer the use of resources. This is necessary for applications that require an execution of different critical GPU accelerated tasks concurrently.

Furthermore, most PBVR methods for unstructured grids do not take final opacity values of rendered volumet-

*philip.voglreiter@student.tugraz.at
†kainz@icg.tugraz.at

ric cells into account. On the one hand, many approaches do not allow this because of inherent problems of the particle generation method. On the other hand, most methods consider particles as completely opaque primitives. Thus, the opacity of a particle is neglected.

Contribution We provide a fast, on the fly method for parallel, real-time particle generation in tetrahedral grids and simultaneous rendering. The proposed method prevents visual patterns such as streaks or clusters in the final images. We also introduce an improved method for particle superimposing. Thereby, we address perceptive issues occurring at different depth levels of the rendered volumes.

2 Previous work

In [2], Avila *et al.* propose an approach of direct volume rendering and define a rendering pipeline for irregular datasets. They refer to the plane-sweep technique, which is widely used to solve geometrical problems. Shirley *et al.* [17] first describe a method of projecting tetrahedrons onto the image plane. The tetrahedrons need to be sorted before projection. Sorting is known to be $O(n\ log\ n)$ for most sorting algorithms, and thus a larger grid size means more computational effort.

Approaches like projected tetras have already been implemented on the GPU [11]. The authors exploit the capabilities of shaders and CUDA to perform depth sorting of the tetrahedrons. Sorting large numbers of tetrahedrons is a time-consuming task and can be inefficient. As alternative approach, Challinger [6] describes a method for ray casting of unstructured grids. Ray casting generates images of a higher quality but shows an $O(n^3)$ complexity in the worst case. However, ray casting offers ways to benefit from modern GPU capabilities as was shown in [21]. Still, the rendering itself requires a high computational effort and is usually too slow for real-time applications.

Point splatting [20] is a method very similar to particle-based approaches. The authors show an efficient way to generate oval splats with low memory cost, but point splatting inherently produces artifacts in the rendering process.

In [16], Sakamoto *et al.* describe a general approach of PBVR. They base their particle generation algorithm on the Metropolis Method [12]. The Metropolis method is a well-known, efficent Monte-Carlo algorithm [13] for random number generation. Generally, the Metropolis method is rather inefficient concerning computational speed. In [15], the authors go deeper into detail of their particle generation method. Also, they consider rendering tetrahedral grids by voxelizing them. Voxelizing a tetrahedral grid can be rather time-consuming, depending on the vertex distribution. Vertices, which are not located exactly at corner points of the rectilinear grid, which characterizes a voxelized volume, need to be interpolated. But the voxel values need to be interpolated again for actual rendering. Interpolation inherently produces erroneous results. Interpolating interpolated values increases the amount of error even further. Pelt *et al.* [19] use a particle-based method to perform illustrative volume rendering. They describe hatching and stippling techniques using particles and also visualize contours of datasets with their method.

3 Particle-based volume rendering

The main idea of PBVR is to construct a dense field of light-emitting, opague particles inside a volumetric dataset. These particles are used to perform object-based rendering by simulating the light emission of particles. Mutual occlusion induced by completely opague particles plays a major role during rendering. Sakamoto *et. al.* [16] describe the basic model in more detail. Generally, PBVR involves two major steps. First, a proper particle distribution inside the volume needs to be generated, which is described in Section 3.1. Second, in Section 3.2 we outline how the particles are projected onto the image plane. In these two sections, we give a detailed description of those two steps as well as some detail on methods to increase the visual performance of the algorithm.

3.1 Particle Generation

In this paper we use a stochastic process to generate the field of particles. It is desirable to generate particles uniformly distributed over the whole volume. This results in images without visual artifacts, namely streaks, holes, or clusters. We split the volume into tetrahedral cells and perform particle generation per cell. This divide and conquer approach has several effects. On the one hand, particle generation is parallelizable. On the other hand, the generated particles do not necessarily resemble a uniform random distribution over the whole volume anymore. Thus, we will show how to treat this situation effectively in the following paragraphs.

Particle Distribution over Cells We consider a maximum number of particles p_{max} for the whole model. This number comprises the maximum amount of particles to be rendered throughout the whole volume. Note that the maximum amount of particles is rarely fully exploited. To accomplish a visually acceptable distribution of particles, we need to determine the amount of particles p_{cell} that each cell may project. We calculate this number by using the proportion of cell volume V_{cell} to the total volume of the grid V_{grid}. This ratio directly describes how many particles of the total quantity we may use for a given cell. Therefore, the number of particles per cell is

$$p_{cell} = V_{cell}/V_{grid} * p_{max}. \quad (1)$$

This can be proven easily for one dimension. Generalization to the third dimension thereafter is trivial, but unfortunately both exceed the length of this paper.

Using this method, we generate one single dense distribution for each cell. What we actually want to achieve is a uniform distribution over the whole volume. Our method of splitting the whole volume into separate sub-volumes shows a statistical benefit. In short, we can distribute the particles over the cells in a way that resembles the distribution of the mean values of the generated particles for different spatial regions. The mean value of the generated particles in a cell – considering a distribution over the whole volume rather than over single cells – is exactly the proportion of the cell volume V_{cell} to the total volume V_{grid}. Thus, we may generate the particles per cell and still statistically achieve a uniform distribution over the whole data space.

Particle Position For the positions of the particles, we use barycentric coordinates on the tetrahedral cells. Barycentric coordinates describe a point that is guaranteed to be within the borders of a given polygon. In case of tetrahedrons, the barycentric notation of a point inside it is

$$P = \alpha * V1 + \beta * V2 + \gamma * V3 + \delta * V4 \tag{2}$$

where V1,V2,V3 and V4 denote the corner points of the tetrahedron and α through δ resemble the barycentric parameters. However, some constraints apply to these parameters. First, each parameter must be greater than zero. Second, all four parameters must sum up to one. Thus, we can rewrite the parameter δ to

$$\delta = 1 - (\alpha + \beta + \gamma) \tag{3}$$

and after replacing δ in Equation 2, the barycentric description of a point results in

$$P = \alpha * V1 + \beta * V2 + \gamma * V3 + (1 - (\alpha + \beta + \gamma)) * V4 \tag{4}$$

This means that we only need to randomly generate parameters α , β and γ for each particle. By using Equation 3, we can calculate δ directly.

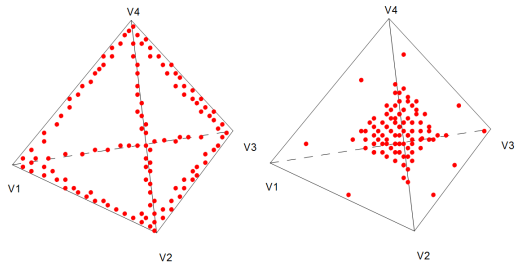


Figure 1: Generated particles clustered near the edges of a tetrahedron on the left and near the center on the right. This effect is created due to using an incorrect distribution

There are several ways to generate barycentric coordinates randomly. Many of those approaches possess statistically correct mean values but still introduce disturbing visual patterns. The most straight-forward way is random generation of all four parameters and dividing them by their sum. This leads to a statistically recorded mean value of 0.25 for each parameter. But the parameters are not statistically independent if generated this way. The following *Definition* shows the computation of the parameters:

Definition 3.1 *Barycentric parameters generated as random numbers over the interval* $(0,1)$ *and their expected values after applying the barycentric summation constraint are given as*

$$E(x) = \frac{0.5}{E(\alpha) + E(\beta) + E(\gamma) + E(\delta)} \tag{5}$$

$$x \in \{\, \alpha, \beta, \gamma, \delta \} \tag{6}$$

However, the particles tend to concentrate in the cell centers, which leads to disturbing visual effects. The border regions of the cell remain very sparse. Another method involves parameter generation within the mentioned constraints. After each parameter is generated, the remaining maximum is updated and used for the generation of the next parameter. This leads to the mean values

$$E(\alpha) = 0.5, E(\beta) = 0.25, E(\gamma) = 0.125, E(\delta) = 0.125 \tag{7}$$

To equalize the distribution, parameters can be shuffled randomly. This circumvents the situation of the first parameter averagely using half of the parameter range and leads to statistically recorded mean values of 0.25 for each parameter. But this method suffers a problem similar to the one of the straight-forward method. The generated stochastic variables are not independent. This approach produces the opposite of the simplistic generation. Cell centers are sparsely covered with particles and cell borders show a strong visual pattern. Both problematic methods of particle generation are illustrated in Figure 1

A method to generate particles with a statistically correct, patternless distribution was introduced by Glassner [7]. The method bases on folding geometry. The barycentric parameters are randomly generated in a parallelepiped, which comprises of the desired tetrahedron and its mirrored counterparts. After generation, the parameters are fitted to the desired tetrahedron. In detail, we first randomly generate the parameters α , β and γ within the range $(0,1)$. In the next step we calculate $sum = \alpha + \beta + \gamma$. If that sum is greater than one, we need to manipulate the generated parameters as we violate the barycentric constraint of parameter summation equaling one. Therefore, we compute $p = 1 - p$ for each parameter. In the final step we calculate $\delta = 1 - (\alpha + \beta + \gamma)$. Figure 2 shows a proper particle distribution achieved by the described method.

Summarizing, the particles are generated uniformly distributed in a parallelepiped. Points which are outside the

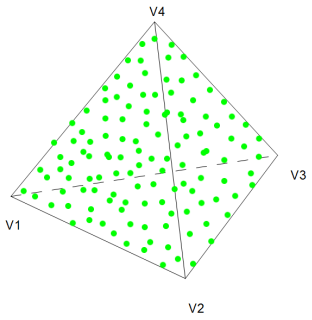


Figure 2: Uniform random distribution of particles over a tetrahedral cell showing no visual patterns

tetrahedron are transformed inside. This method generates a patternless uniform random distribution of coordinates within the constraints of barycentric parameters.

Particle Scalar Value So far we are able to generate a cloud of uniformly random distributed particles. The scalar value corresponding to each particle is easy to calculate. We already know the geometric influence of each corner point to a given particle, namely the barycentric coordinates. We can reuse those parameters to calculate the scalar value of a particle as

$$s = \alpha * s1 + \beta * s2 + \gamma * s3 + (1 - (\alpha + \beta + \gamma)) * s4, \quad (8)$$

where s1 through s4 denote the scalar values of the corner points.

Particle Emission Probability In section 3.1 we describe a uniform particle distribution over the whole grid. Simply projecting all generated particles would lead to a high density of particles hitting the screen regardless of cell opacity. So we need to thin out the particle field. As we still want to avoid patterns within the rendered images, we do this stochastically.

Using the scalar value and a transfer function, we first determine the opacity that a particle would anticipate. Based on this calculated opacity, we use the rejection method [13] to decide whether a particle is emitted or not. Generally speaking, the determined opacity of a particle op, which is in the interval $(0,1)$ describes the emission probability. Corresponding to each particle we next generate a stochastic variable x within the interval $(0,1)$ on the real line. Now we perform an emission check, i.e. if x is smaller than op, the particle is accepted and emitted. Otherwise, the particle is discarded. When applying this method, cells with a mean opacity close to 1 emit almost all of their generated particles, while cells with a low opacity end up with sparse particle coverage.

3.2 Particle projection and Image Generation

Particle projection involves two steps. First, the screen space location of the particle needs to be determined. We need the virtual camera parameters and volume transform to achieve this. Second, a color value needs to be assigned to each particle.

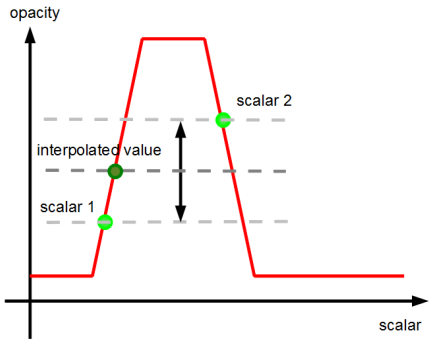
Projection from Object Space to Image Space By using the modelview-projection matrix of the viewing camera, we determine the image-space position of each emitted particle. Further, we calculate its distance to the camera. This involves a simple matrix - vector multiplication. Should two particles hit the same fragment on the image plane, the one closer to the camera is chosen to be displayed. The particle with a bigger distance is discarded. This way, no depth sorting of any kind is necessary before or during rendering. We only need to compare the depth values of subpixels. This approach is comparable to z-buffering. Therefore, it is necessary to create two buffers, one for color, and one for depth.

Transfer function Looking up the corresponding color of a given scalar value in a transfer function is possible at three different stages of our approach. The first possible lookup can happen before projection. This means assigning the corresponding color to the corner points of the cells. To calculate the color of a particle, one needs to interpolate the colors of the corner points. This method is called pre-classification. The next possible lookup may happen during projection of a particle. In post-classification, as opposed to pre-classification, the scalar values of the corner points are interpolated. Assigning a color value to a particle is done by applying the transfer function to the interpolated scalar value. Both pre- and post-classification have the same memory footprint, due to the fact that the RGBA value of a pixel uses the same amount of memory as a floating point scalar value. Also, the computational effort for both methods is roughly the same. Our approach offers one more possibility. One can store the scalar value until subpixel aggregation and perform the lookup for the final interpolated scalar of a pixel.

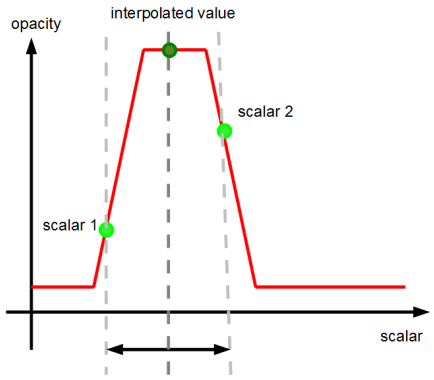
We use post classification per default because of the advantage in interpolation accuracy. Figure 3 illustrates the difference between pre- and post-classification.

3.3 Spatial superimposing

To achieve a higher degree of projection accuracy as well as a translucent appearance of the grid we use spatial superimposing. This means that each pixel is subdivided into several subpixels. The subpixel level l describes the amount of subpixels per pixel, where the number of actual subpixels equals $l * l$. The particles are thereby projected

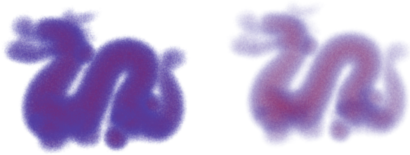


(a) pre-classification



(b) post-classification

Figure 3: The difference in interpolation between different classification strategies. Figure (a) shows pre-classification where final opacity and color values are interpolated. In (b) scalar values are interpolated using post-classification and the application of the transfer function is performed using the interpolated scalar.



(a) l=3 (b) l=6

Figure 4: Stanford Dragon rendered with 60 million particles per frame, subpixel level 3 (a) and subpixel level 6 (b) on a resolution of 1200x800 pixels

onto a subpixel instead of a whole pixel. This increases the amount of particles actually reaching the image plane. Finally, the resulting pixel is calculated by averaging the values of the subpixels.

Translucency Translucency is controlled by two parameters. Firstly, the amount of generated particles influence how opaque the volume seems to be. The more particles are generated, the more subpixels are hit. Secondly, the subpixel level increases or decreases the level of transparency. When there are more subpixels in total, there are also more subpixels, which are not hit by particles. Hence, in the aggregation process we find more empty subpixels, decreasing the mean opacity of a superpixel and making the color appear to be brighter.

Unfortunately, increasing the subpixel level also increases the amount of used memory on the GPU drastically. Therefore, a proper subpixel level considering the tradeoff between accuracy and feasibility needs to be found for each GPU model. Also, the number of particles and the subpixel level need to be balanced for reaching the desired level of transparency and computational performance. This balance needs to be adjusted individually for each graphics card and desired volume translucency.

Particle Depth Enhancement Simple averaging of subpixels results in an unwanted visual effect. The original particle position and thereby the distance to the camera is not taken into account. Thus, averaging treats each subpixel as the same, resulting in an equal visualization of particles disregarding their distance to the camera. The effect is best compared to front face culling in mesh rendering. It might lead to a wrong depth perception while viewing rendered volumes. While it is hardly perceivable on static images, the viewer might become aware of it when rotating or panning the volume. Perceivably, the rendered volume does not respond to transformation as expected. To circumvent this effect, we use the already present depth information of displayed particles.

In detail, we analyze the current depth of each subpixel z_{curr} and record minimum z_{min} and maximum depth z_{max} for a pixel. We then calculate the depth range. Next, we calculate a depth ratio ζ, considering the gap to the maximum value.

$$\zeta = (z_{max} - z_{curr})/(z_{max} - z_{min}) \tag{9}$$

Using ζ as factor for the RGBA values of subpixels, we achieve a linear differentiation of particles respective to their depth values. Particles with a higher distance to the viewer have a smaller impact on final pixels than particles close to the camera.

4 Implementation

We implemented our method using CUDA 3.2 [1], C++ and OpenGL using a NVidia GForce GTX 470 graphics card. We use GLUT to create an OpenGL viewer and to provide necessary camera controls.

4.1 Preprocessing of Datasets

We preprocess the datasets using the Visualization Toolkit [10]. We iterate over all cells and determine their type. If we encounter a non-tetrahedral cell we tetrahedralize the cell if possible. In the next step we convert the VTK Unstructured Grid to a VBO containing a simple data structure. Each tetrahedron consists of four vertices plus their corresponding scalar value. We then map this VBO to the GPU for final processing.

4.2 GPU Preparation

We need to store the transfer function for the scalar value lookup and we also need to transfer the current model-view-projection matrix to the device. Cuda random number generation needs an array of states, which we allocate and setup on the GPU once. Furthermore, we allocate different buffers in the GPU's memory. We create two buffers for storing the projected particles in subpixels, one for color and one for depth. Finally, we use a pixelbuffer for the final image, which we can map to a texture on the screen.

4.3 Cuda Kernels

Projection Kernel The computing grid of this kernel is linear. We use a blocksize of $256 \times 1 \times 1$ and calculate the grid size to be $\#tetras/256 \times 1 \times 1$. Each thread handles one tetrahedron. First we calculate the number of particles to be generated by the current thread. For each particle, we perform the following steps. First, we generate the barycentric parameters according to Section 3.1. Next we calculate the particle's scalar as described in Section 3.1 and perform an opacity-only lookup for this scalar. We test this scalar against a random number, which we generate to determine whether it is projected or not. This process is depicted in the flow chart of Figure 5. If this test succeeds, we calculate the position of the particle and finally project it via multiplication with the model-view-projection matrix. This generates screenspace coordinates, which we look up in our buffer. If the corresponding subpixel is already covered by a particle, we perform a depth check to determine whether we need to overwrite the current subpixel and the depth value accordingly. A visualized flow of this process can be seen in Figure 6.

Superimposing Kernel We define a block size of $16 \times 16 \times 1$ and a grid size of $windowwidth/16 \times windowheight/16 \times 1$. Each thread handles one block of

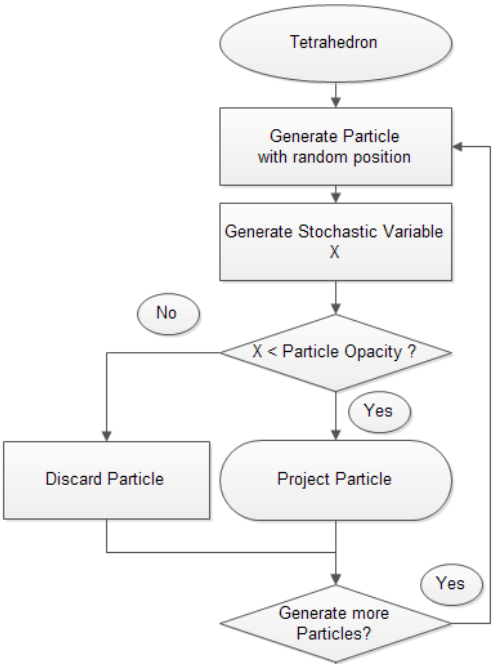


Figure 5: Flow chart depicting particle generation

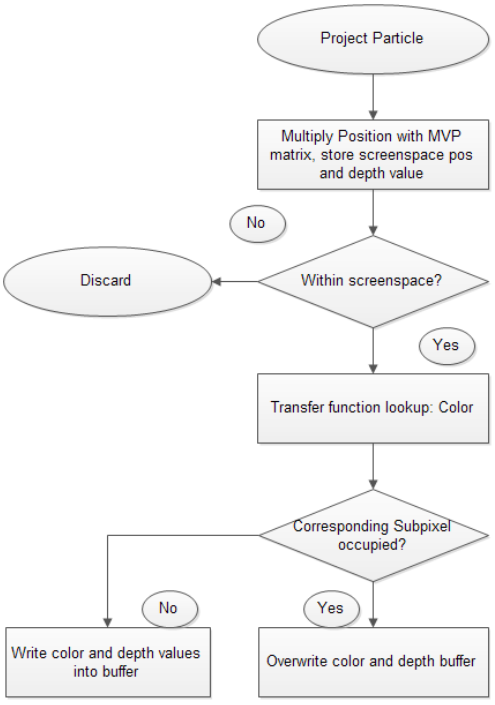


Figure 6: Flow chart depicting particle projection

subpixels corresponding to the number of subpixels per pixel. In a preliminary step we determine the minimum and maximum depth value in the current subpixel block as well as the distance between those extremes. We consecutively look up each subpixel, multiply it by the value described in Section 3.3, and add them up. Finally we divide the summation by the total amount of subpixels per

pixel and store the calculated value in our pixelbuffer.

Gaussian Smoothing As an optional step for improving visual results of our renderer, we post process the images we generate. Using the same computation grid as above, we apply a 3×3 gaussian blur to the pixelbuffer and store the result in the final pixelbuffer, which is displayed. Figure 7 shows a comparison of two images, one of which was smoothed, and the other was not.

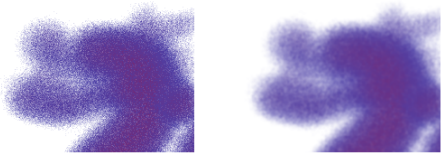


Figure 7: Stanford Dragon with 60 million particles, subpixel level 3. The figure on the left shows the unsmoothed version. The figure on the right contains the same area of the volume, but with a 3x3 gaussian kernel applied to it.

4.4 Displaying Rendered Images

To finally display the rendered image, we use a screen-sized texture quad and map the filled final pixelbuffer to it. As all our data structures are already OpenGL-conform, there is no further processing neccessary.

5 Results

For testing we use several volumetric datasets. One of our main test volumes is the Stanford Dragon volume [18]. We have tetrahedralized a voxel volume to obtain a regular tetrahedron grid. This grid consists of 588245 tetrahedral cells. Another test volume is a completely unstructured grid obtained from a simulated radio frequency ablation in liver tissue as described in [4] and [9]. This test volume consists of 55527 cells.

Figure 4 shows two images of the Stanford Dragon rendered with a maximum amount of 60 million particles on subpixel level 3 and 6. A comparison of those images exposes the importance of balancing the amount of particles with the subpixel level to obtain the desired transparency of the volume.

In Figure 8 we show the decrease in performance with increasing number of particles and increasing subpixel levels tested with the Stanford Dragon volume. From a certain point, neither increasing the subpixel level nor increasing the particles by one step drastically decreases the frame rate. Taking the high amount of cells of this dataset into account, the recorded data show that our approach scales well with a high amount of particles and a high subpixel level.

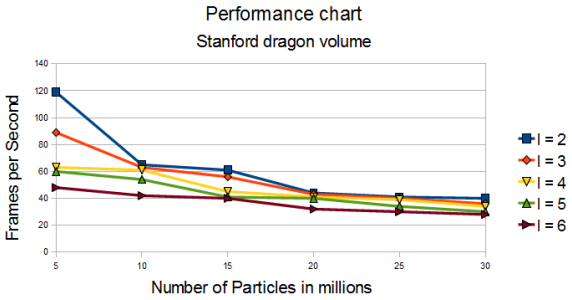


Figure 8: Frames per second for different amounts of particles and subpixel levels while rendering the Stanford Dragon volume. Lines depict the flow of performance for fixed subpixel levels.

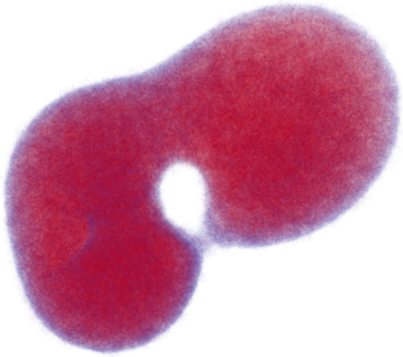


Figure 9: Radio frequency ablation simulation, subpixel level 3 and 6 million particles per frame at 55 fps

Figure 9 shows the radio frequency ablation simulation dataset. The dataset was constructed from a finite element simulation, using an unstructured grid. We have preprocessed this grid to split up non-tetrahedral cells. The volume itself shows probability of cell death during a radio frequency ablation. The saturated, red areas in the center have a high probability of cell death while the blueish border regions are more likely to survive the treatment. Hinted on the left and obvious in the center, the viewer can see veins penetrating the area, working as a heat sink and thereby increasing probability of cell survival. This is depicted in the hazy border regions as those areas have lower opacity, and thereby a lower amount of particles to be emitted.

6 Conclusion

We have shown that our method is able to render millions of particles per second in real-time. This is mainly possible achieved by our particle generation process. We generate the particles per-frame and in real-time, and take care of proper distribution over the volume. Further, our approach offers capabilities to render arbitrary volumetric data structures as most data sets can easily be converted to a tetrahedral structure. The opposite, correct voxelization of unstructured data, is a lot harder to achieve.

In the future, this approach might be expanded to enable rendering multiple volumes. Also, the approach could be modified to distributedly render very high resolution images for large displays. A detailed benchmark test and comparison against other volume rendering approaches is also part of future work.

References

[1] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[2] R. Avila, Taosong H., Lichan H., A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang. VolVis: a diversified volume visualization system. In *Visualization, 1994., Visualization '94, Proceedings., IEEE Conference on*, pages 31 –38, CP3, Oct. 1994.

[3] I. Babuska. Generalized Finite Element Methods : Main Ideas , Results , and Perspective. *Security*, 1(1):67–103, 2004.

[4] T. Bien, G. Rose, and M. Skalej. FEM modeling of radio frequency ablation in the spinal column. In *Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on*, volume 5, pages 1867 –1871, oct. 2010.

[5] T.P. Caudell and D.W. Mizell. Augmented reality: an application of heads-up display technology to manual manufacturing processes. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, volume ii, pages 659 –669 vol.2, jan 1992.

[6] J. Challinger. Scalable parallel volume raycasting for nonrectilinear computational grids. In *Proceedings of the 1993 symposium on Parallel rendering*, PRS '93, pages 81–88, New York, NY, USA, 1993. ACM.

[7] A.S. Glassner. Generating random points in triangles. In *Graphic Gems*, pages 24 – 28. Academic Press, 1990.

[8] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[9] D. Haemmerich, S. Tungjitkusolmun, S.T. Staelin, Jr. Lee, F.T., D.M. Mahvi, and J.G. Webster. Finite-element analysis of hepatic multiple probe radio-frequency ablation. *Biomedical Engineering, IEEE Transactions on*, 49(8):836 –842, Aug. 2002.

[10] Kitware Inc. The Visualization Toolkit. `http://www.vtk.org`, February 2012.

[11] A. Maximo, Marroquim R., and Farias R. Hardware-Assisted Projected Tetrahedra. *Computer Graphics Forum*, 29, Issue 3:903–912, 2010.

[12] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[13] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[14] S. Roettger and T. Ertl. A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *Volume Visualization and Graphics, 2002. Proceedings. IEEE / ACM SIGGRAPH Symposium on*, pages 23–28, Oct. 2002.

[15] N. Sakamoto, J. Nonaka, K. Koyamada, and S. Tanaka. Particle-based volume rendering. In *Visualization, 2007. APVIS '07. 2007 6th International Asia-Pacific Symposium on*, pages 129 –132, Feb. 2007.

[16] Naohisa Sakamoto, Jorji Nonaka, Koji Koyamada, and Satoshi Tanaka. Volume Rendering Using Tiny Particles. In *Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on*, pages 734 –737, Dec. 2006.

[17] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. In *Proceedings of the 1990 workshop on Volume visualization*, VVS '90, pages 63–70, New York, NY, USA, 1990. ACM.

[18] Stanford University. The Stanford 3D Scanning Repository. `http://graphics.stanford.edu/data/3Dscanrep/`, February 2012.

[19] R. van Pelt, A. Vilanova, and H. van de Wetering. Illustrative Volume Visualization Using GPU-Based Particle Systems. *Visualization and Computer Graphics, IEEE Transactions on*, 16(4):571 –582, July-Aug. 2010.

[20] F. Vega-Higuera, P. Hastreiter, R. Fahlbusch, and G. Greiner. High performance volume splatting for visualization of neurovascular data. In *Visualization, 2005. VIS 05. IEEE*, pages 271 – 278, Oct. 2005.

[21] Changgong Zhang, Ping Xi, and Chaoxin Zhang. CUDA-Based Volume Ray-Casting Using Cubic B-spline. In *Virtual Reality and Visualization (ICVRV), 2011 International Conference on*, pages 84 –88, Nov. 2011.

Rapid Visualization Development based on Visual Programming Developing a Visualization Prototyping Language

Benedikt Stehno*
Supervised by: Eduard Gröller, Martin Haidacher

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Vienna / Austria

Abstract

In this paper we introduce a dataflow visual programming language (DFVPL) and a visual editor for the rapid development of visualizations. It enables users with only little programming experience to develop custom visualizations. With this programming language, called OpenInsightExplorer, users can develop visualizations by connecting graphical representations of modules rather than writing source code. Each module represents a part of a processing step in the visualization pipeline. Modules are designed to function as an independent black box and they start to operate as soon as data is sent to them. This black box design and execution model allows to reuse modules more frequently and simplifies their development.

The usability of the programming language was evaluated by implementing two example visualizations with it. Each example originates from different areas of visualization (scientific and information visualization), therefore demanding different data types, data transformation tasks and rendering.

Keywords: visual programming, rapid prototyping, dataflow programming

1 Introduction

Visualization is a scientific research field which deals with developing computer aided techniques for visually representing large quantities of data. These techniques transform data into meaningful visual images, that should allow people to gain insight and help to interpret the data. Some of these techniques are interactive and allow to analyze the data sets in an interactive manner. Every visualization follows the concept of the visualization pipeline (see Figure 1) [3, 4, 10]. To develop a custom visualization a user needs to implement the stages of the visualization pipeline. The visualization pipeline consists of the following successive processing steps:

Data acquisition: In the first step the user defines the data source from which the data should be loaded. The data may get read out of special formatted files, databases or various other sources like simulations or real time measurements. Additionally this step often contains a *data analysis* process. The dataset is prepared for visualization in this step, e.g. by interpolating missing values, applying a smoothing filter or correcting erroneous measurements.

Filtering: Filtering is a user centered step. The user selects the portions of data he/she wants to be visualized. For example, a user selects data out of a certain time range, which should be visualized.

Mapping: The focused data gets mapped to geometric primitives (e.g. points and sprites) and their attributes (e.g. color, position). The focused data gets transformed to geometric data in this process stage.

Rendering: The final step of the pipeline transforms the geometric data into the resulting image, providing the *visualization output*.

Users who want to rapidly develop a visualization for certain data can use visualizations packages. Most of them are specialized to a certain field of visualization making it rather complicated to extend them with custom needed functionality. Such extensions can actually only be developed by users with significant programming knowledge. Moreover developers need to be familiar with the programming language the visualization application is written in.

In contrast OpenInsightExplorer supports visual programming which even non programmers can learn in a short timespan. The framework contains modules which can be combined in a graphical editor to a custom visualization pipeline. Only missing functionality needs to be implemented by developing new modules and adding them to the framework. Since modules in OpenInsightExplorer are designed to work as independent black boxes this can be easily achieved.

The next section deals with the state-of-the-art of dataflow visual programming languages and provides an

*benedikt.stehno@gmail.com

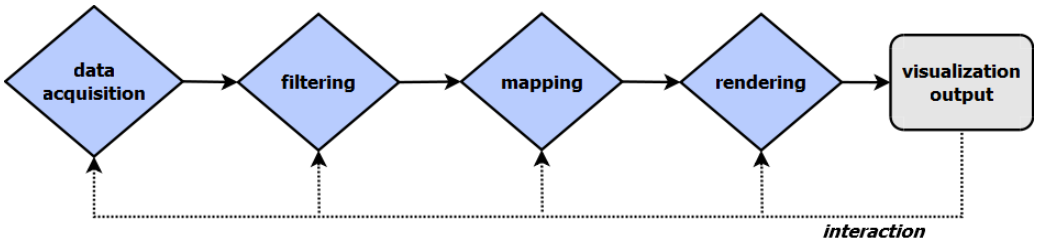


Figure 1: Visualization pipeline: It describes all possible processing steps from the data acquisition to the final visualization output. Since interaction plays a crucial role in visualization a user may interact with all processing steps of the pipeline in order to generate the desired visualization output.

introduction to their underlaying paradigms, visual programming and dataflow programming. In Section 3 an overview of the OpenInsightExplorer framework is given and its features are described in detail. The implementation of the framework is discussed in Section 4. The results that could be achieved by implementing two example visualizations with the framework are described in Section 5, followed by a conclusion including a discussion about future work in Section 6.

2 State of the Art

OpenInsightExplorer is based on two paradigms, the visual programming paradigm and the dataflow programming paradigm. These two paradigms were merged together, resulting in the family of *dataflow visual programming languages (DFVPL)*, to which OpenInsightExplorer belongs to.

Visual programming languages (VPL) allow users to program by manipulating or arranging graphical elements rather than writing textual source code. Users arrange or combine graphical symbols, following the specific syntax rules of a language.

Visual programming languages can be designed to work on a higher abstraction level than their textual counterparts using graphical metaphors [8]. This gives users the ability to work with them in a more intuitive way. Often they reach such an abstraction level that no prior programming experience or knowledge is required to express or design programs. Hence they often are used for *End User Development*, where users can create, modify or extend parts of a software without any significant knowledge about programming.

All of the programming languages presented in this section follow the concept of *boxes and arrows* [1, 5, 6]. Boxes represent independent modules which are connected by arrows in a graphical editor. The modules exchange data over these connections. They perform calculations or tasks as soon as they have received all necessary data for the execution. This principle is called the *dataflow execution model* [7, 13, 15]. In this model modules can be executed in parallel when they have received all necessary data for an execution.

Programming becomes in dataflow visual programming languages the task of connecting modules to a graph or network. This concept was implemented for example by the following state-of-the-art dataflow visual programming languages:

One of the first commercial DFVPLs was LabVIEW [9, 12]. LabVIEW is still in development and several versions of the platform have been released. With this software users can build virtual instruments by connecting different function nodes within a block diagram by drawing wires. It has been shown that large projects can be developed faster with a visual programming language in comparison to traditional text based programming languages [2]. LabVIEW became an industrial success and its benefits made it popular among researchers.

The DFVPL concept was also adopted for visualization purposes. OpenDX (Open Data Explorer) [16] is a cross-platform scientific data visualization software. It can deal with different kinds of data such as scalar, vector or tensor fields. A noteworthy feature of OpenDX is that it supplies GUI modules for interaction. With them the user is able to manipulate various aspects of the visualization with graphical user elements. Some of these, so called interactors, were developed to be smart and data driven. For example, sliders can automatically determine the minimum and maximum value(s) of the dataset, setting its boundaries appropriately.

Another example of a DFVPL for visualization purposes is MeVisLab [14, 19]. It is a very specialized visualization package for medical imaging and processing. It integrates VTK (Visualization Toolkit) [22, 20] modules in addition to its own ones to provide a wide range of specialized visualization modules.

But DFVPLs can also be found in the field of rendering and graphics processing. Quartz Composer [18] developed by Apple allows users to develop graphical rendering applications, e.g. for music visualization or as system screen saver. This framework can only handle built-in data types.

The presented state-of-the-art DFVPLs are highly optimized for their specific purposes. None of them was designed to serve as a language for scientific and information visualizations programming. E.g. Quartz Composer does not allow users to introduce custom data types, which proved to be a key feature for information visualization

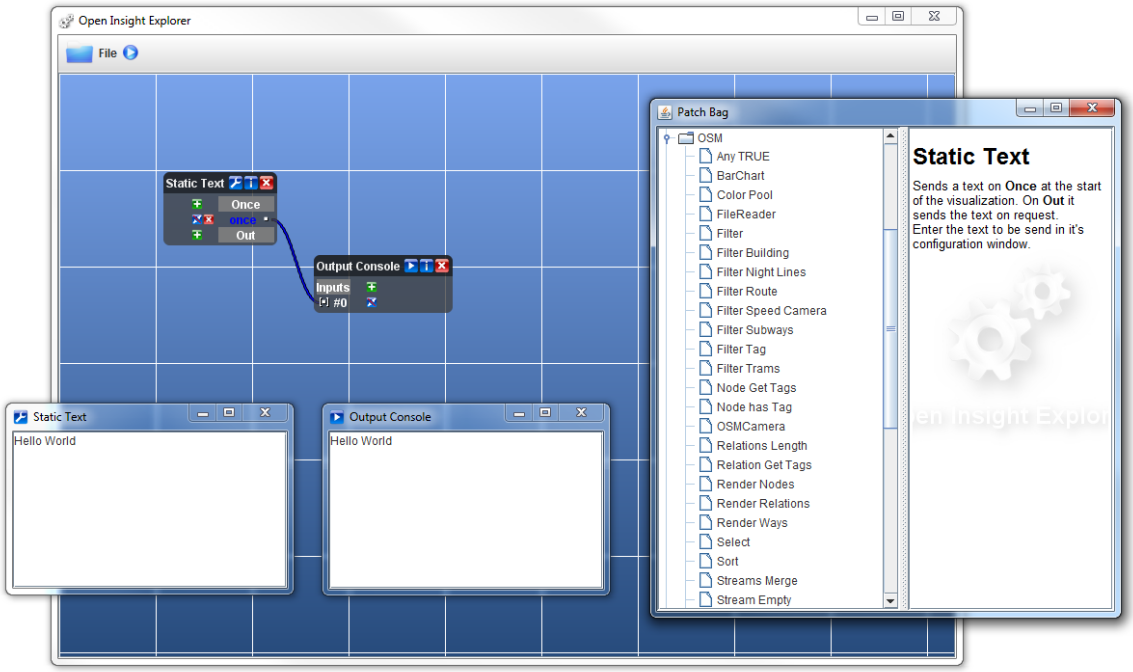


Figure 2: A screenshot of the visual editor of OpenInsightExplorer. The editor's main window is depicted in the center and on the right side the patch repository (*Patch Bag*) is shown.

(see Section 5.2). Also OpenDX, which is a scientific visualization language, is incapable of supporting information visualization features. Additionally extending OpenDX with new functionality seems to be a more complex task in comparison to OpenInsightExplorer, which was specially designed for rapid prototyping and to be easy extendable with new modules (see Section 4).

The following section provides an introduction to OpenInsightExplorer and describes the features it provides to fit the needs as a general purpose visualization DFVPL.

3 OpenInsightExplorer

OpenInsightExplorer allows users to program their custom visualizations visually. Users simply connect graphical representations of modules in a visual editor rather than writing source code. Each module represents a part of a certain stage of the visualization pipeline (Figure 1). Figure 2 depicts a screenshot of the visual editor of OpenInsightExplorer.

There are modules that cover the step of *data acquisition*, for example a module that loads data from a file. Other modules may transform this data to geometric primitives. This occurs in the *mapping* stage of the pipeline. Connecting multiple individual modules with certain functionality together results in building a custom visualization pipeline. The user-defined connections express paths on which the data flows from one module to the next.

Figure 3: A screenshot of a *patch* with an *input port* and an *output port*.

The modules are called *patches* in the OpenInsightExplorer framework. Figure 3 depicts a screenshot of a simple patch. They operate as independent *black boxes*. That means that the user does not need to know precisely how they work. It is only necessary to know what they do. Since every stage of the visualization pipeline exchanges data with its preceding and/or succeeding stage, patches need to exchange data with each other as well. They have so called *input ports* and *output ports* (see Figure 3). Through the input ports a patch receives data. It processes the data and passes its results to another patch through its output ports.

To create visualizations with OpenInsightExplorer, users only need to find patches with the desired functionality and connect them in the visual editor of the framework (as depicted in Figure 2). Patches can be found in the patch repository window entitled *Patch Bag*. They are sorted in a tree structure by their functionality. When a user selects a patch in the repository, information about the patch is displayed in the right section of the *Patch Bag*

window. Selected patches can be dragged in the main editor's window where they can be connected. Visualizations developed with the OpenInsightExplorer are called *compositions*. Using this simple visual programming concept allows users with little programming experience to program custom visualizations [9].

Like all other DFVPLs OpenInsightExplorer supports automatic parallelization of the execution of its modules. Patches can be executed in parallel as soon as they have received all necessary data for an execution, since it uses the same *dataflow execution model* principle on which every DFVPL relies on. Important and partially unique features of OpenInsightExplorer in comparison to the state-of-the-art languages presented in this paper are listed below:

Platform Independence: The framework is written in *Java*, which is a platform independent programming language. Many platforms and architectures support runtime environments for Java (JRE) and can run software written in Java.

Growing Ports: The growing ports mechanism of OpenInsightExplorer is a unique feature which the presented state-of-the-art programming languages do not provide. It allows to add and remove ports dynamically to a patch while editing a visualization. Figure 4 shows a simple example of the mechanism. E.g. a patch that determines the maximum of a set of numbers should be flexible with respect to the number of operands of the function - hence the number of input ports.

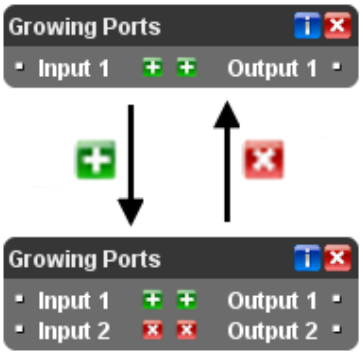


Figure 4: Illustrating the growing ports mechanism. Ports can have *add* and/or *remove* icons which will trigger the mechanism.

Streams: Instead of sending only individual data tokens between patches, OpenInsightExplorer implements the concept of *token streams* [15]. Patches can have special *stream* ports which enable them group data together to a stream. A stream consists of a start token, an ordered sequence of data and a token which will signal the end of a stream. Streams can also be embedded into another stream, which is a big improvement over flat arrays. These streams within streams are called sub streams.

Port Trees: Ports can be organized in trees. Also labels can be added to port trees (Figure 5). This allows to structure the input and output ports of a patch, and to add and remove ports dynamically.

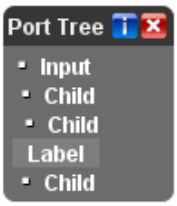


Figure 5: Ports are organized in a tree structure.

Generic Ports: To make patches more flexible, OpenInsightExplorer features generic port types. Patches can have ports, which are not assigned to a certain data type. As soon as they are connected, they can adapt their data type to the type of the connection partner. They can change their data type dynamically. This feature allows to implement patches, which can operate on any desired data type and can be used more frequently.

Patch GUI: Developers can place GUI elements of a patch in three different locations. Patches can have a *running* GUI which is a window that will be visible during the runtime of a visualization. For example the *Renderer* patch providing an OpenGL render surface uses the running GUI window for output purposes. The second possibility to add a GUI is the *configuration* GUI. This window will only be visible during editing a visualization. It is useful to display GUI elements that configure the behaviour of a patch. The third location is the *bound* GUI. It is directly visible between the input and output ports of a patch (Figure 6).

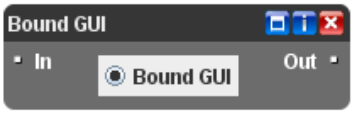
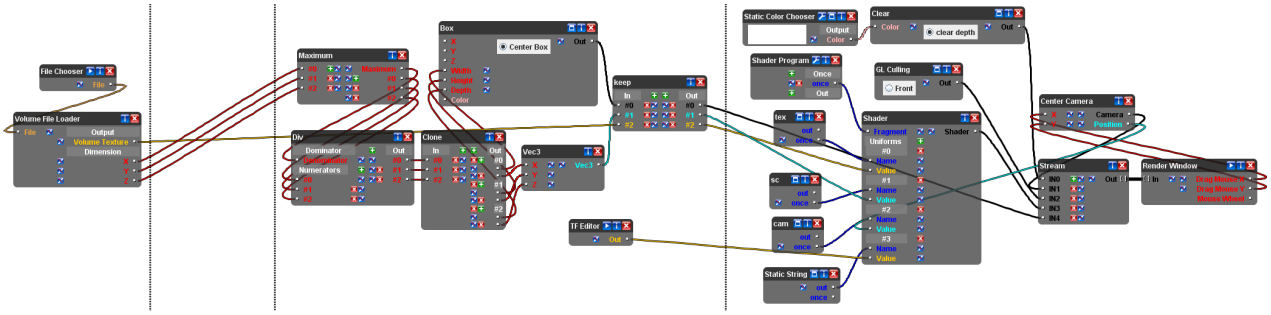


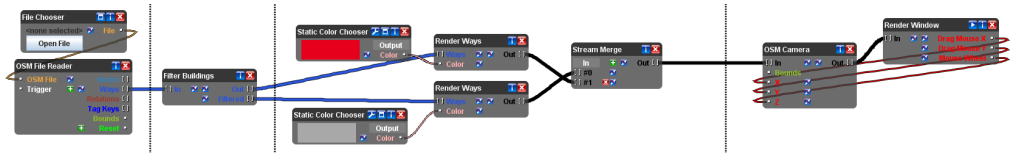
Figure 6: A patch with a *bound* GUI.

Custom Data Types: Unlike some state-of-the-art DFVPLs OpenInsightExplorer allows users to introduce new data types. Ports can be constructed with any arbitrary data type developers of a patch may desire. This is in our opinion a very important feature because visualizations can be build upon very different data types (e.g. volumetric data, data structures that represent graphs).

These classes can contain methods and functions in addition to the data. For example, a class that represents a graph can have a method which returns the names of



(a) Volume renderer composition.



(b) OpenStreetMap visualization composition.

Figure 7: Two example visualization compositions. They are split into following successive stages of the visualization pipeline: *data acquisition*, *filtering*, *mapping* and *rendering*

the nodes of the graph sorted. Furthermore such a class could implement many different interfaces and therefore represents multiple data types at once. For example, a graph class can implement two interfaces at once: one represents an undirected graph and the other one a directed graph.

Delegating Patches: The exchange of classes containing functions enables the development of patches that follow the *delegation* pattern. A patch can call a function of a previously received helper object and therefore delegates certain needed functionality to it. This can greatly enhance the usability of patches.

Type-safety: Ports in the framework support a type-safety mechanism. Every port of a patch is constructed for a certain data type (with the exception of generic ports which were discussed before). It can only send or receive a certain data type it was assigned to. Whenever a user tries to connect two patches in the visual editor OpenInsightExplorer verifies if the data types of the input port and output port are compatible. The color of the name of a indicates hints the data type the port was constructed for.

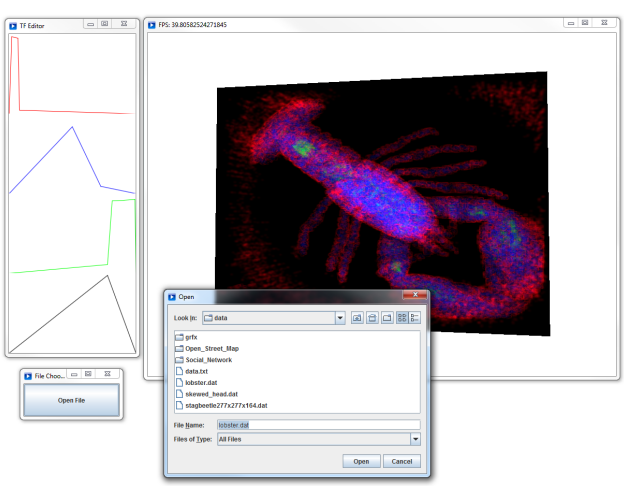
The next section deals about general architecture of the framework and how some of the previous mentioned features are implemented.

4 Implementation

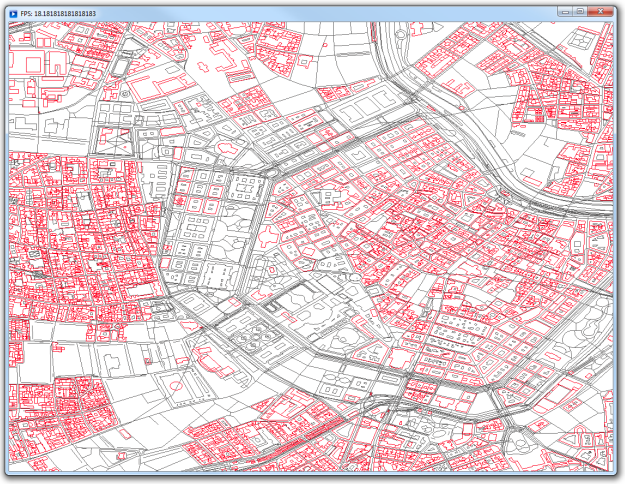
The framework and the visual editor is written in the platform independent programming language Java. OpenInsightExplorer loads platform dependent native libraries at runtime, which makes it possible to port the framework to different operating systems and hardware architectures. However the current version only supports the operating system Windows, since needed native libraries for OpenGL rendering (Jogl [11]) are distributed only for that operating system with the framework. But porting to other platforms / operating systems should be a feasible task.

OpenInsightExplorer can be extended with new features by developing new patches and adding them to the framework. This process was designed to be easy to accomplish, since patches are designed to operate as independent black boxes. To create a new patch, users only need to implement a certain *Patch* interface (see Listing 1). Its design is inspired by the Java's Applet interface. It contains methods to initialize *init()* and *reset()* a patch and the methods *start()* and *stop()* for its execution. In addition it declares methods that return references to the GUI elements of a patch and its input and output port trees. A new patch only gets compiled once and its binary *class* file copied into the patch entitled sub-folder of the framework. At every startup of OpenInsightExplorer the framework scans this folder and displays all found patches in the patch repository.

To send and receive data, patches can instance input or output port objects and add them to their appropriate port tree. Port objects can only be assigned to a certain data type, since it is a class with a generic type parameter. Many different *callback functions* can be registered to ports. These functions enable the implementation of the growing port mechanism and generic ports. They are called depending on different events. E.g. a generic port calls a specific callback function as soon as the user tries to connect this generic port to another port in the visual editor. Developers can implement code into this callback



(a) On the left is the transfer function editor depicted and in front a file open dialogue. The right window is the rendering output window, displaying the volume visualization.



(b) All ways are rendered in gray and buildings in red.

Figure 8: Screenshots of the example visualizations: the volume renderer (a) and the OpenStreetMap visualization (b).

function that will adapt the patch so it can handle the data type of the connection partner or refuse the connection attempt. The same principle is applied to the growing port mechanism (see Figure 4). The add and remove icons trigger different callback function in which a developer can implement code that will add or remove one or more ports to/from the port trees.

```
public interface Patch {

  public void init();
  public void reset();
  public void start();
  public void stop();

  public String getInfo();
  public String getName();

  public void load(Serializable o);
  public Serializable save();

  public Port getInputPorts();
  public Port getOutputPorts();

  public JPanel getBoundGUI();
  public JFrame getConfigurationGUI();
  public JFrame getRunningGUI();

}
```

Listing 1: The *Patch* interface.

The implementation of the framework is hidden from patch developers by applying the *cheshire cat* programming pattern [21]. They are only confronted with functions or methods which are truly necessary for developing patches and cannot (accidentally) access the framework internals. Applying this pattern to the framework ensures that patches can work only as absolute independent black boxes and enforces the interchangeability of patches.

5 Results

To evaluate the usability of the OpenInsightExplorer framework two example visualizations were implemented with it. The first example is a volume renderer, which comes from the field of scientific visualization. To test the frameworks information visualization capabilities, the second example is a collection of different visualizations of the OpenStreetMap project. The example visualizations demand different data types, data transformations and rendering techniques.

5.1 Volume Rendering

The first example visualization which was implemented to evaluate the framework is a simple volume renderer based on raycasting. Figure 8(a) depicts a screenshot of the running visualization. The goal was to use only general purpose patches of the framework whenever possible for the volume visualization. Only two custom patches, the *Volume File Loader* and the *Transfer Function Editor*, had to be implemented in addition. The first one loads the volume data from a file and converts it to a 3D texture. The support for other volume file formats can be easily achieved by developing other patches for those formats. Figure 7(a) illustrates the composition of the volume renderer. The example uses GPU hardware acceleration for image generation. This is achieved by using GLSL fragment shader programs, which implement the ray sampling and composition functions. This example composition contains only 21 patches in total.

5.2 OpenStreetMap Visualization

The second example visualizes data from the *OpenStreetMap (OSM)* [17] project. OpenStreetMap is a collaborative project to create a free editable map of the world. Maps from OpenStreetMap contain information about highways, buildings, public transport and much more. For this example visualization a map of the city of Vienna was used. It was extracted from the OpenStreetMap database.

OpenStreetMap maps can be exported to XML-files. These files are built on only three simple elements: *node*, *way* and *relation*. Each element may have an arbitrary number of properties (*tags*) which are key-value pairs (e.g. highway=primary). Since OpenInsightExplorer allows users, as a feature, to introduce arbitrary data types, these elements can be mapped to specially developed classes.

The visualization shows all streets and buildings of Vienna. Figure 8(b) depicts a screenshot of the visualization: street are visualized in gray and buildings in red. The corresponding composition contains only 10 patches in total (see Figure 7(b)). This example demonstrates the capabilities and the benefits of the usage of individually implemented data structures to create patches. The big advantage is that the user can achieve the desired result with very few lines of code. The example also emphasizes the fact that re-usability is highly given. If a patch is already implemented it does not require much effort for minor modifications to re-use it for another purpose.

6 Limitations and Conclusions

Like most other dataflow languages, OpenInsightExplorer is prone to dataflow network deadlocks. It does not introduce new features for deadlock prevention or recognition to the field of dataflow language research. OpenInsightExplorer follows a coarse grained dataflow approach. This means that the modules are rather complex and such deadlocks seldom occur.

It does not support any kind of structured programming, which nearly all current visual dataflow programming languages do. Also OpenInsightExplorer provides only a basic debugging support in comparison to other state-of-the-art languages. Extending the framework with more sophisticated debugging tools and structured programming support should be a very feasible task.

Despite the existing drawbacks of the framework, OpenInsightExplorer contains unique features in comparison to the presented state-of-the-art languages, like the growing port mechanism and generic ports. Both mechanisms enable the development of more flexible and reuseable modules. Developers can use and introduce arbitrary data types to the framework. Many other existing visual dataflow programming languages are not capable of this.

OpenInsightExplorer cannot be used as a universal tool for non-programmers for developing arbitrary visualizations. But users with programming experience can benefit from the framework. They are able to implement all missing modules and can reuse already existing ones. This speeds up the development process and allows to rapidly prototype rather simple visualizations. Nevertheless OpenInsightExplorer implements features, which could bring great benefits to other visual dataflow languages. They are worthwhile to be adopted by current state-of-the-art languages, which may not possess them, to improve their usability.

References

[1] K. Arvind and D.E. Culler. *Dataflow architectures*, pages 225–253. Annual Reviews Inc., Palo Alto, CA, USA, 1986.

[2] E. Baroth and C. Hartsough. *Visual programming in the real world*, pages 21–42. Manning Publications Co., Greenwich, CT, USA, 1995.

[3] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[4] E. H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, pages 69–75, Washington, DC, USA, 2000. IEEE Computer Society.

[5] A. L. Davis and R. M. Keller. Data flow program graphs. *Computer*, 15:26–41, 1982.

[6] J. B. Dennis. First version of a data flow procedure language. In *Programming Symposium, Proceedings Colloque sur la Programmation*, pages 362–376, London, UK, 1974. Springer-Verlag.

[7] J. B. Dennis and D. P. Misunas. A preliminary architecture for a basic data-flow processor. *SIGARCH Computer Architecture News*, 3:126–132, 1974.

[8] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev. Meta-design: a manifesto for end-user development. *Communication of the ACM*, 47:33–37, 2004.

[9] T. R. G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A 'cognitive dimensions' framework. *Journal of visual languages and computing*, 7:131–174, 1996.

[10] M. Haidacher. *Information-based Feature Enhancement in Scientific Visualization*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2011.

[11] Jogl. http://kenai.com/projects/jogl/pages/Home. Accessed: 2011-02-06.

[12] W. M. Johnston, J. R. P. Hanna, and R. J. Millar. Advances in dataflow programming languages. *ACM Computing Surveys*, 36:1–34, 2004.

[13] R. Karp and R. Miller. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM Journal*, 14:359–370, 1966.

[14] MeVisLab. http://www.mevislab.de/. Accessed: 2010-11-09.

[15] J. P. Morrison. *Flow-Based Programming, 2nd Edition: A New Approach to Application Development*. CreateSpace, Paramount, CA, 2010.

[16] OpenDX. http://www.opendx.org/index2.php. Accessed: 2011-03-12.

[17] OpenStreetMap. http://www.openstreetmap.org. Accessed: 2011-03-17.

[18] Quartz Composer. http://developer.apple.com/graphicsimaging/quartz/quartzcomposer.html. Accessed: 2011-02-06.

[19] J. Rexilius, J. M. Kuhnigk, H. K. Hahn, and H. O. Peitgen. An application framework for rapid prototyping of clinically applicable software assistants. In Christian Hochberger and Rdiger Liskowsky, editors, *GI Jahrestagung (1)*, volume 93 of *LNI*, pages 522–528. GI, 2006.

[20] W. J. Schroeder, K. M. Martin, and W. E. Lorensen. *The visualization toolkit: an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., 2 edition, 1998.

[21] H. Sutter. *Exceptional C++ Style: 40 New Engineering Puzzles, Programming Problems, and Solutions*. Pearson Higher Education, 2004.

[22] Visualization Toolkit. www.vtk.org. Accessed: 2011-06-01.

Poster Session

Robust Volume Segmentation using an Abstract Distance Transform

Márton Tóth
Dávid Dvorszki
*Supervised by: Balázs Csébfalvi**

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest / Hungary

Abstract

In this paper, a novel segmentation technique is introduced that is based on an abstract distance transform. The user can select a voxel as a seed point, from which the other voxels can be reached along different paths. For each voxel, we determine a path that is of minimal cost. Taking the density of the seed point as a reference, the cost of a path is calculated as an aggregate deviation of the densities corresponding to voxels visited along the given path. After having the cost of the cheapest path assigned to each voxel, a simple thresholding is used to obtain a segmentation mask. We demonstrate that this approach is competitive to the popular level set segmentation, but can be implemented more efficiently on recent Graphics Processing Units (GPU). Previously, using high-level shader languages for direct programming of the graphics pipeline, the implementation of different segmentation methods was difficult. However, new languages such as OpenCL or CUDA provide a more flexible environment for GPGPU (General Purpose computing on the GPU) programming. We show that, utilizing this new technology, our algorithm can be easily realized.

Keywords: Segmentation, GPGPU programming, Volume rendering

1 Introduction

Segmentation is a fundamental task in 3D medical image processing [13]. The goal is to identify different organs or tissues in the data. Usually a binary classification of the voxels is not possible, instead a probability is obtained that expresses how much the given voxel belongs to a certain organ. Furthermore, a simple thresholding [16] of the density values generally cannot be applied for soft tissue segmentation, since due to the noise contained in the data it often leads to under or oversegmentation. Therefore, many more sophisticated segmentation algorithms have been proposed that utilize secondary information, such as gradient or curvature [13, 11, 7, 15], distance or connectivity [13, 14, 11], or a priori anatomical information [3]. Most of these methods are computationally expensive, therefore their traditional CPU implementation is rather inefficient.

Recently, the conventional GPUs are more and more used as general-purpose coprocessors exploiting their parallel computing capacity. Originally, the GPUs were designed for fast incremental image synthesis, which is the core of many computer graphics applications. GPUs are in fact optimized for rendering huge polygonal meshes. The standard computer graphics pipeline includes vertex processing, rasterization, fragment (or pixel) processing, and compositing. The first technological milestone was reached when the vertex and pixel processing became programmable, since it allowed the redefinition of the standard pipeline for general-purpose computation [18]. However, using direct shader programming for the implementation of complex segmentation techniques was still cumbersome. For example, volumetric data represented by texture maps could not be read and written at the same time, and the number of instructions in the vertex and pixel shaders was also limited. Therefore, several rendering passes were required for mapping the segmentation process onto the graphics hardware.

OpenCL and CUDA represent the second milestone in GPGPU programming [17]. Using these languages for developing GPU applications, the programmer can abstract from the graphics pipeline considering the GPU as a general multi-processor architecture. In this paper, we show that this higher flexibility can be very well exploited in GPU-based segmentation.

In Section 2, we briefly review the previous methods that are related to our work. In Section 3, the GPU implementation of the popular Level Set (LS) method is described. Our new segmentation technique called Abstract Distance Transform (ADT) is introduced in Section 4. In Section 5, ADT is compared to LS on a brain segmentation task. Finally, in Section 6, the contribution is summarized.

*cseb@iit.bme.hu

2 Related Work

Medical image segmentation has a wide literature. Although there exist general segmentation techniques [13], practical methods are often designed and optimized for a very specific segmentation task. In this section, we overview only those previous techniques that are the most closely related to our method.

The simplest segmentation technique is thresholding [16]. It requires only two parameters, a lower and a higher threshold. If the intensity of a voxel is between these two threshold values then it belongs to the segmentation mask. Thresholding works fine if the given material is well defined by an intensity interval like the bone in a CT scan, for instance. Soft tissue structures, however, are much more difficult to segment. Especially in MRI data sets, the data values are not so coherent as in CT data because of the higher noise-to-signal ratio and the so called global signal fluctuation [12]. Therefore, the connectivity information plays a crucial role in the segmentation process. Region-growing techniques try to find voxel regions that are connected and show a nearly homogeneous density distribution. This is an iterative approach started from a user-selected seed point. In each iteration step, the boundary voxels of the current segmentation mask are investigated and it is decided whether their neighboring voxels should be added to the mask or not. Variants of the region-growing method differ in the criteria that are used for this decision [11, 14, 13]. For example, edge detection operators are usually applied to find the material boundaries [11]. In practice, pure region-growing often leads to so called segmentation leakage [7], where the binary mask unexpectedly leaks through the boundaries of the organs to be segmented. This problem can be somehow handled by starting a region growing from anti seed points, which fills those voxels that should not be added to the segmentation mask. However, an appropriate placement of the anti seed points might be difficult or time-consuming for the physician. In order to complete the segmentation without significant user intervention, the level set method was proposed [15]. This approach is also based on region growing, but it defines additional constraints on the boundary surface of the segmentation mask. Usually, the iteration steps are performed in such a way that the total curvature of the boundary surface is kept under a reasonable level. As we compare our new graph-based method to the level set technique, in Section 3, we describe its GPU implementation in detail.

There exist other graph-based segmentation techniques that are commonly used in practice, such as intelligent scissors [10] or the graph-cut method [2]. Both of them consider the image as a weighted graph. With intelligent scissors, the user can define a cut in a 2D image by giving two points and the algorithm calculates a path between them trying to follow edges as far as possible. The weights are calculated by edge detection methods and the path is constructed by a shortest path method. However, intelligent scissors cannot be easily extended to 3D [4]. In the graph-cut technique, the user can define a foreground and a background point and the algorithm calculates a border between them separating the two regions based on the max-flow min-cut theorem. The weights can be estimated by using region and edge properties. This approach can be applied in 3D as well. Our method is based on aggregate deviation calculation and uses the Bellman-Ford algorithm to determine a shortest path between pairs of voxels.

3 GPU-Accelerated Level Set Segmentation

The LS algorithm consists of an initialization and an iterative region growing. In the initialization step, the user specifies a spherical level set inside the region of interest. Based on the position and the radius a 3D signed distance map is created. The voxels in this distance map represent the distance of the nearest surface point. The sign indicates whether the given voxel is inside or outside the spherical surface. During the region growing, in fact, the distance values are modified in each iteration step, so the zero-crossing level set surface is evolving until the desired segmentation mask is obtained. The distance values are modified depending on the voxel intensities and the curvature of the level set surface.

The data term of the distance update equation is defined as follows

$$D(\vec{p}) = \varepsilon - |I(\vec{p}) - T|, \tag{1}$$

where $\vec{p}$ is the voxel position, $I(\vec{p})$ is the voxel intensity at $\vec{p}$, and T is the target intensity. Parameter ε is set by the user during initialization. The data term is shown in Figure 1 as a function of the intensity.

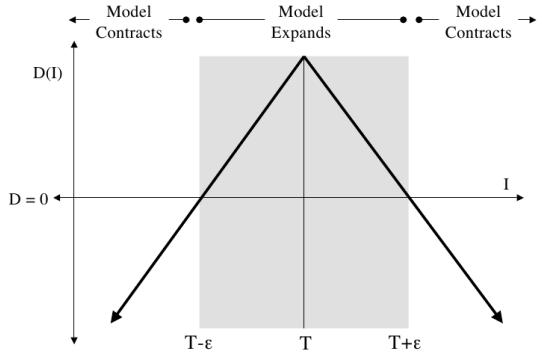


Figure 1: The data term depending on the voxel intensity [7].

The curvature term can be defined as

$$C(\vec{p}) = \nabla \cdot \frac{\nabla \phi(\vec{p})}{|\nabla \phi(\vec{p})|}, \tag{2}$$

where $\phi(\vec{p})$ represents the distance to the nearest surface point at position $\vec{p}$. Using Equations 1 and 2, the distance update equation is constructed as

$$\frac{\partial \phi(\vec{p})}{\partial t} = |\nabla \phi(\vec{p})| \Big[\alpha D(\vec{p}) + (1-\alpha) C(\vec{p}) \Big], \qquad (3)$$

where α is also a user defined parameter of the algorithm. Without the curvature term, the algorithm would be a simple flooding based on intensity, but using curvature calculations helps avoiding the leakage of the segmentation in ambiguous regions.

Since the update process locally modifies the distance values $\phi(\vec{p})$, it is necessary to recalculate $\phi(\vec{p})$ from time to time in order to consistently represent the distance to the evolved surface. The Fast Iterative Method (FIM) [6] is a convenient way to accomplish this, since it can be parallelized as well. Using FIM, the voxels are divided into three sets: source voxels, active voxels and far-away voxels, and their distance values are iteratively updated until they converge. Figure 2 illustrates the propagation of the distance calculation frontwave, where the black grid points are the source voxels with fixed distance value, the blue grid points are the active ones being updated, and the white points are considered as far-away voxels.

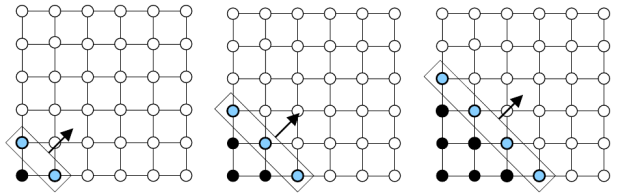


Figure 2: FIM frontwave propagation [6].

3.1 Implementation

The equations introduced previously need to be discretized in time and space as well. The implementation of the data term calculation is the easier. Since the user sets all parameters during initialization, we need to calculate the data term values only once. If we store these precalculated values in an array, we can access them later during the update steps. However, the curvature term calculation must be approximated. The applied approximation method is known as "difference of normals" [9].

Having both terms calculated, we need to update the distance values according to Equation 3. This needs to be discretized in time, thus we need to perform the following update steps:

$$\begin{aligned}
\phi(\vec{p})^{(t+1)} &= \phi(\vec{p})^{(t)} + \Delta\phi(\vec{p})^{(t)} \\
&= \phi(\vec{p})^{(t)} + \Delta t |\nabla \phi(\vec{p})^{(t)}| \Big[\alpha D(\vec{p}) + (1-\alpha) C(\vec{p}) \Big] \\
&= \phi(\vec{p})^{(t)} + \Delta t |\nabla \phi(\vec{p})^{(t)}| \left[\alpha D(\vec{p}) + (1-\alpha) \nabla \cdot \frac{\nabla \phi(\vec{p})^{(t)}}{|\nabla \phi(\vec{p})^{(t)}|} \right],
\end{aligned}$$

It is important to apply an appropriate "bravery" factor, to ensure accuracy and avoid having large steps that could ruin the segmentation. In this implementation we set $\Delta t = 5$ and the whole $\Delta\phi(\vec{p})^{(t)}$ can have a value of maximum 0.6.

The distances are recalculated after every third iteration of the level-set update. A distance recalculation consists of two parts. First, the positive distance values are set to infinity and classified as far-away voxels, and the ones with negative (or zero) distance are preserved and classified as source voxels. During the second part of the FIM, distance values are updated in several steps. Before an update step, we find active voxels, that is, a voxel having a source voxel as one of its first neighbors becomes active. Then the distance of each active voxel is updated based on the other active and source voxels in its neighborhood. A voxel becomes a source voxel if its distance has converged (has not changed significantly during the last update step). Therefore, in the next update step, we may find other active voxels.

3.2 Optimization

There are several ways to optimize the segmentation process. First of all, it can be implemented on the GPU exploiting its parallel computational power in the data term and curvature term calculations, and in the distance recalculation as well. Using narrow-band and sparse-field methods, we can even limit the calculations to the voxels that are near the surface[8].

Our implementation is based on a similar optimization. Since the computational bottleneck of the algorithm is the distance recalculation, the volume is divided into cubes of size $4 \times 4 \times 4$. Before each FIM update step, we can determine which cube contains voxels close to the surface. In these voxels accurate distance values are required. Since the update process of the FIM is performed multiple times, these cubes need permanent updating to decide whether they contain active voxels or not. This checking does not produce any significant overhead, but the distance recalculation becomes four to five times faster without the need to change the storage scheme.

4 Abstract Distance Transform

Our new method is designed to avoid segmentation leakage, but without expensive curvature calculations. The key idea is to calculate an abstract distance of each voxel from a user-defined seed point. Assume that a path of length N between the seed point $\vec{p}_0$ and an arbitrary voxel passes through voxels located at $\vec{p}_i$, where $i \in \{1,2,...,N\}$. The cost of the path is defined as

$$\sum_{i=1}^{N} |I(\vec{p}_i) - I(\vec{p}_{i-1})|, \qquad (4)$$

where $I(\vec{p})$ is the intensity of the voxel at position $\vec{p}$. In each voxel, we calculate the cost of the cheapest path from

the seed point. These abstract distance values represent the accumulated deviation along the path and not some kind of approximation of the Euclidean distances as in case of traditional distance transforms. Having the cheapest paths determined for each voxel, a 3D Abstract Distance Map (ADM) is obtained. In order to produce a binary segmentation mask, a simple thresholding is applied on the ADM.

The calculation of the ADM is, in fact, the classical problem of finding the shortest path in a weighted graph. Given two nodes u and v in a graph G, the length of a directed path from u to v is the sum of the weights of the edges along the path. The shortest path from u to v is a directed path which has minimal length among the possible paths from u to v in G. Now we can consider the distance $d(u,v)$ between u and v. This distance is 0 if $u = v$, it is ∞ if there is no path from u to v in G, otherwise the distance can be measured as the length of the shortest path from u to v. In general $d(u,v) \neq d(v,u)$ because the path is directed. This distance measurement is not always meaningful. Consider a situation when u and v is placed on a directed cycle, which has negative sum of weights. In this case we could achieve an arbitrarily small distance between the two nodes by going through the cycle multiple times. So it is required, that the given graph G does not contain a directed cycle with negative sum of weights.

Assume that $G = (V,E)$ is a directed graph, $c : E \to R^+$ is a cost function with non-negative values, and $s \in V$ is the source node. Our goal is to determine the value of $d(s,v)$ for each $v \in V$. In our application, V is the set of all the voxels, E is the set of edges between the adjacent voxels, and c is the difference between the intensities of two adjacent voxels.

The shortest path can be found using the classical Bellman-Ford algorithm [5]. This algorithm is described by the following pseudo code:

```
Step 1: initialize distances

  for each node v in V

    if v is seed
    then v.distance := 0
    else v.distance := infinity

Step 2: relax distances repeatedly

  for each node v in V

    for each edge uv in E

      u := uv.source
      v := uv.destination
      if u.distance + uv.weight < v.distance
      then v.distance := u.distance + uv.weight
```

This algorithm runs in $O(|V| \cdot |E|)$ time, where $|V|$ is the number of nodes and $|E|$ is the number of edges. $|E|$ is proportional to $|V|$ because of the geometrical structure of the volume, so the overall computational complexity is $O(|V|^2)$.

4.1 GPU Implementation

Because of the high amount calculations, it is worthwhile to implement our method on the GPU. An iteration of the distance relaxation can be calculated in a parallel way. We store the distance of each voxel in an array, and for each array element calculate the new distance by considering the distances to the adjacent voxels as the difference between the intensity values. In each iteration, it is decided whether the distance values have to be modified. If the distance value of a voxel is already minimal, the algorithm does not change it, but the adjacent voxels can potentially get a new, smaller value. If the distances do not change anymore, the iteration is terminated.

An iteration is calculated by the following code:

```
/*
input:  input array of distances
result: output array of distances
c:      termination condition
dimx, dimy, dimz: dimensions of the volume
*/
__global__ void cuda_BellmannFord(
  unsigned short* input,
  unsigned short* result, unsigned short* c,
  int dimx, int dimy, int dimz)
{
  long i = __umul24(blockIdx.x, blockDim.x)
           + threadIdx.x;
  long z = i / (dimx * dimy);
  long y = (i % (dimx * dimy)) / dimx;
  long x = (i % (dimx * dimy)) % dimx;

  if(x < dimx - 1 &&
     y < dimy - 1 &&
     z < dimz - 1 &&
     x > 0 && y > 0 && z > 0)
  {
    unsigned short u =
      input[x + y * dimx + z * dimx * dimy];
    unsigned short vv =
      tex3D(volumeTexture, x, y, z);

    unsigned short tmp = USHRT_MAX;

    unsigned short v;
    unsigned short w;

    int dd = 1;

    for(int dz = -dd; dz < dd+1; dz++)
      for(int dy = -dd; dy < dd+1; dy++)
        for(int dx = -dd; dx < dd+1; dx++)
        {
          if((z + dz) >= 0 && (z + dz) < dimz &&
             (y + dy) >= 0 && (y + dy) < dimy &&
             (x + dx) >= 0 && (x + dx) < dimy)
          {
            if(dx != 0 || dy != 0 || dz != 0)
            {
              v = input[(x + dx) + (y + dy) *
                dimx + (z + dz) * dimx * dimy];
              w = abs(vv - tex3D(volumeTexture,
                x + dx, y + dy, z + dz));
              tmp = v + w < tmp ? v + w : tmp;
            }
          }
        }
```

```
    if(tmp < u) c[0] = 1;

    result[x + y * dimx + z * dimx * dimy] =
      tmp < u ? tmp : u;
  }
}
```

In worst case, $|V|-1$ iteration steps have to be performed (the longest path in G can not be longer then $|V|-1$, an iteration updates each voxel, so the aggregated number of the started threads is $O(|V|^2)$), but the iteration can be terminated if the distance map does not change. The segmentation mask is obtained by an appropriate thresholding of the obtained distance map.

5 Results

We have tested our method for brain segmentation from CT and MRI data, and compared the results to that of the classical LS method. Figure 3 shows the obtained segmentation masks rendered by direct volume visualization. Note that the two algorithms provide almost the same results. However, our method is significantly faster to evaluate on the GPU. The running times measured on an NVIDIA GT335M graphics card are summarized in Table 1.

5.1 Accuracy of ADT

To evaluate the accuracy of our method, we compared our segmentation results with fifteen manually segmented MRI images. These segmentations were created by professional physicians and the data were provided by the IBSR project of the Center for Morphometric Analysis at Massachusetts General Hospital [1]. We used the conventional overlap metric for the comparisons, which is defined as $A/(B+C-A)$, where A is the number of voxels that are classified to belong to the segmentation mask by both the expert and the algorithm, B is the number of voxels assigned to the mask by the expert, while C is the number of voxels assigned to the mask by the algorithm. As a reference, the IBSR project provides results of other segmentation techniques and also compares the segmentations of two different experts. The accuracy measurements are summarized in Table 2.

6 Conclusion

In this paper, we have introduced a new and robust method for segmenting noisy CT and MRI data. Compared to the popular LS segmentation, our method provides similar results for a significantly lower computational cost. Using a fast GPU implementation, the calculation of our ADM is an order of magnitude faster that the GPU-accelerated iterative region growing based on the LS approach.

	CT Brain	MRI brain
Resolution	$256 \times 256 \times 159$	$256 \times 256 \times 60$
LS	2790 sec	1265 sec
ADT	151 sec	34 sec

Table 1: Running times of the brain segmentation.

	Average overlap	Description
ADT	0,562	based on 15 images
Expert	0,854	based on 4 images
Other	0,544	based on 20 images and 6 methods

Table 2: Overlap of different techniques.

So far we have tested our method only on a brain segmentation task. In our future work, we would like to make experiments on kidney and liver segmentation as well.

Acknowledgements

This work was supported by the project TÁMOP-4.2.2.B-10/1–2010-0009 and OTKA 101527. The first author is a grantee of the Öveges József Scholarship funded by GE Healthcare Hungary.

References

[1] The internet brain segmentation repository. `http://www.cma.mgh.harvard.edu/ibsr/`.

[2] Yuri Y. Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *International Conference on Computer Vision*, pages 105–112, 2001.

[3] G.E. Christensen, S.C. Joshi, and M.I. Miller. Volumetric transformation of brain anatomy. *IEEE Transactions on Medical Imaging*, 16(6):864–877, 1997.

[4] L. Grady. Minimal surfaces extend shortest path segmentation methods to 3D. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2):321–334, 2010.

[5] G. Ivanyos, L. Rónyai, and R. Szabó. *Algoritmusok*. TYPOTEX ELEKTRONIKUS KIADÓ KFT., 2008.

[6] Won-Ki Jeong and Ross T. Whitaker. A fast iterative method for a class of Hamilton-Jacobi equations on parallel systems. Technical Report UUCS-07-010, University of Utah, April 2007.

[7] A. Lefohn, J. Cates, and R. Whitaker. Interactive, GPU-based level sets for 3D brain tumor segmentation. In *Medical Image Computing and Computer Assisted Intervention*, pages 564–572, 2003.

[8] A. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):422–433, 2004.

[9] A. Lefohn and R. Whitaker. A GPU-Based, Three-Dimensional Level Set Solver with Curvature Flow. Technical Report UUCS-02-017, University of Utah, December 2002.

[10] Eric N. Mortensen and William A. Barrett. Interactive segmentation with intelligent scissors. In *Graphical Models and Image Processing*, pages 349–384, 1998.

[11] T. Pavlidis and Y.T. Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):225–233, 1990.

[12] M.P. Persoon, I.W.O. Serlie, F.H. Post, R. Truyen, and F.M. Vos. Visualization of noisy and biased volume data using first and second order derivative techniques. In *Proceedings of IEEE Visualization*, pages 379–385, 2003.

[13] D. L. Pham, C. Xu, and J. L. Prince. Current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–337, 2000.

[14] J. B. T. M. Roerdink and A. Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41:187–228, 2000.

[15] James Albert Sethian. *Level set methods and fast marching methods: Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1999.

[16] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–165, 2004.

[17] L. Szirmay-Kalos and L. Szécsi. General purpose computing on graphics processing units. In A. Iványi, editor, *Algorithms of Informatics*, pages 1451–1495. MondArt Kiadó, Budapest, 2010. http://sirkan.iit.bme.hu/˜ szirmay/gpgpu.pdf.

[18] L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.

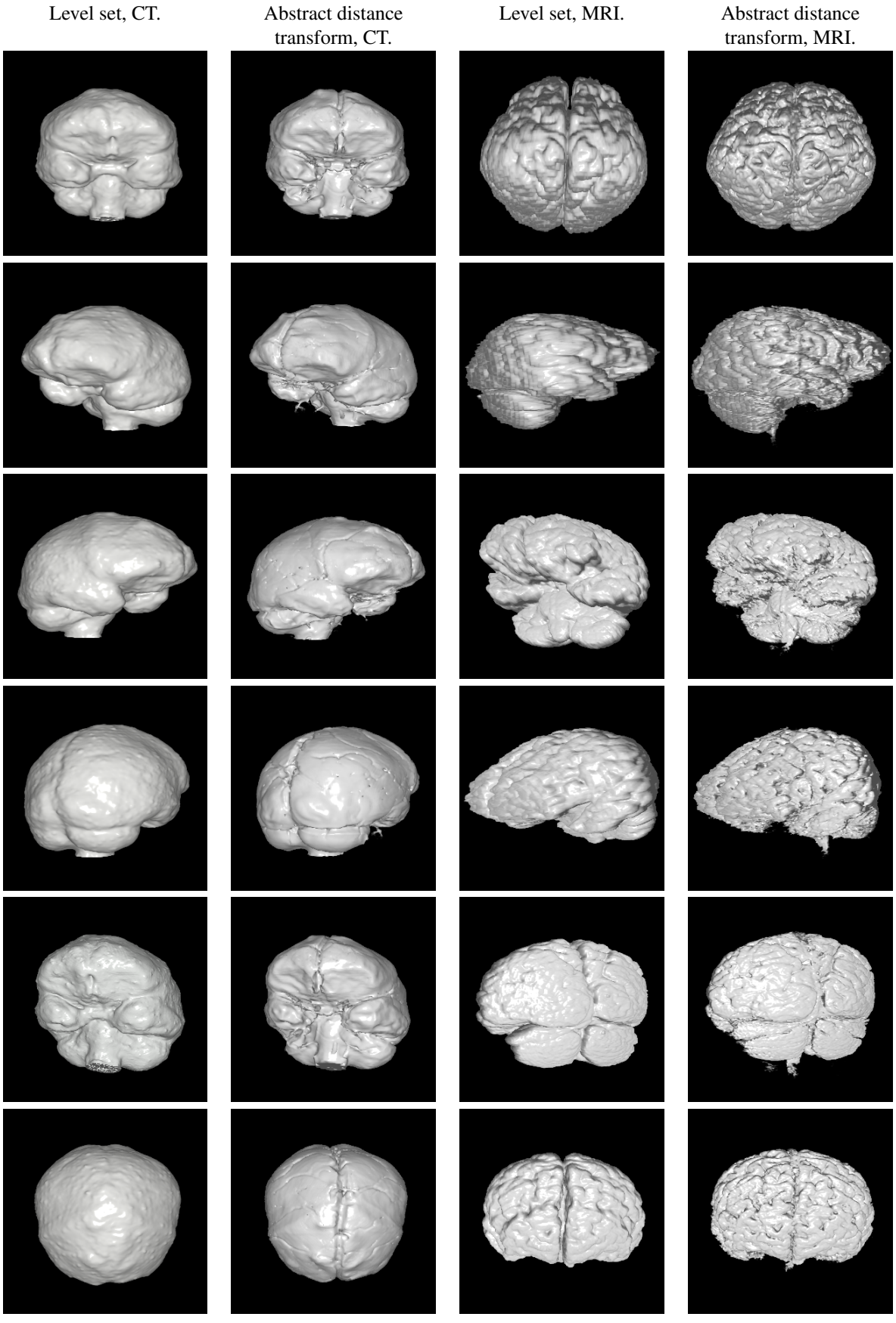


Figure 3: Brain segmentation from CT and MRI data using our abstract distance transform method and the classical level set method.

Deformation of skeleton based implicit objects

Sondre Langeland Hisdal*
Supervised by: Julius Parulek†

Institute of Informatics
University of Bergen
Bergen / Norway

Abstract

In this paper we present a precise contact modeling environment for skeleton based implicit objects. To render the scene composed of these implicit objects, we have implemented the state-of-the-art raycasting algorithm, called marching points, on GPU using CUDA. Further, we introduce how to interactively deform the implicit objects when they collide. To achieve this we studied several ways to deform the objects. We implemented two well-known approaches, where we also proposed a new method created as combination of both approaches. Both these approaches as well as our method are described in this paper.

The implicit objects are implemented as distance surfaces. The field function for these only depend on the distance from the skeleton, and are easy to evaluate. We have achieved support for deformations of objects based on point and line skeletons.

Keywords: Implicit objects, Deformations, GPU, CUDA, Raycasting

1 Introduction

The goal of this project was to implement a precise contact modeling environment for skeleton based implicit objects. This work is an extention of a previous work with convolution surfaces, where convolution of skeleton primitives and an implicit kernel function was used to model merging of implicit objects. Figure 1 shows two spherical objects, made from point-skeletons, merging together. As shown in the figure you can see the objects stretch and merge together when they are brought closer to eachother.

In this project we were not interested in merging of objects, but the deformation of the objects when they collide. Implicit objects can be used to model organic structures and it is interesting to see how these behave when they intersect. It is neccessary to have a fast GPU based deformation of implicit objects if this is going to be interactive. By representing the objects with skeletons we save a lot of memory compared to using meshes. We have implemented support for point and line skeletons for this project. We looked at several ways to achieve deformation. We have implemented two well-known techniques, as well as introduced a new technique which is a combination of both techniques.

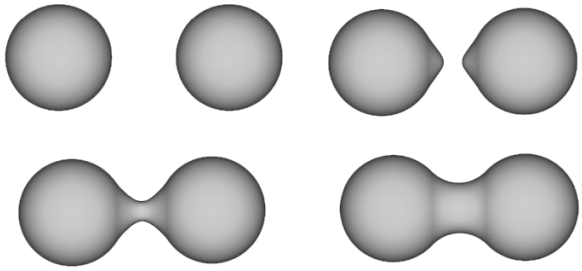


Figure 1: Figure shows two surfaces of point-skeletons merging together.

To achieve an interactive rendering we have implemented a raycasting algorithm called marching points on GPU. Calculation of deformations is very expensive and it needs to be parallelized to be interactive. The marching points algorithm is easily parallelized and well suited for rendering of implicit objects.

The rest of the paper is structured as follows. In Section 2 we discuss related work. In section 3 we describe the modelling and deformation in detail. In section 4 we describe the marching points algorithm. In section 5 we describe implementation and show results. Finally we conclude and talk about future work.

2 Related Work

2.1 Skeleton based implicit objects

Skeleton based implicit objects are objects defined by a skeleton primitive, like a point or a line, and some implicit function. The implicit function creates a object around the skeleton. We are on the surface of the object when the field function equals some iso-value, $f(p) - iso = 0$, where p is a position in space. By evaluating the field function from any position we can tell if the position is inside or outside the object. There are two main approaches to construct

*shi015@student.uib.no
†parulek@gmail.com

an implicit surface from a skeleton, distance surfaces and convolution surfaces. Distance surfaces use the distance from a point to the closest point on the skeleton when calculating the field function. Convolution surfaces integrate the contribution from all points on the skeleton when evaluating the field function. We have used distance surfaces for this project.

2.2 Deformation

For theoretical background for the deformations we have looked at several papers [1, 2, 3, 4, 5] about deformable objects. We have in particular looked at two of these papers[4, 2] for this project, and have implemented the techniques described in these. To our knowledge this has not been done on GPU before, and very little work has been done in this area the last years.

3 Deformations

Deformations should occur when two objects collide. To achieve this a deformation term is added to the field function around the intersection. We will have a surface when

$$f_i + deform(f_j) = iso \tag{1}$$

When looking at object i we calculate the field function of that object and add the deformation term caused by object j on object i. As a part of this project two different approaches for modeling deformations were implemented. We will describe each of these later in this section, but first we will look at how collision detection is done, since the same approach is used for both deformation techniques.

3.1 Collision detection

Before we can add a deformation term to the field function we need to check if the two objects actually intersect. We have done this by checking if the closest middle point between the two skeletons is inside both of the objects. This is easy to do for point skeletons. For point skeletons we just find the middle point between the two skeletons and check if that is inside both. For line skeletons it is a bit more difficult. The way it have implemented, we first find the shortest line between the two line skeletons, and then check if the middle of the line is inside both object. In figure 2 the shortest line segment between the two line segments AB and CD, is the line segment CE. If the objects defined by line segment AB and CD are intersecting, the middle point of CE, point F in the figure, will be inside both objects. For intersection between point and line skeletons we find the closest point on the line to the point and check the middle point of the line going from that point of the line to the point skeleton. This only works when the objects have the same width. If the skeletons have different widths we can not check the middle point, but the same basic approach can be used. First we find the shortest line between the skeletons and then instead of finding the middle we find the point that corresponds to the widths of the objects.

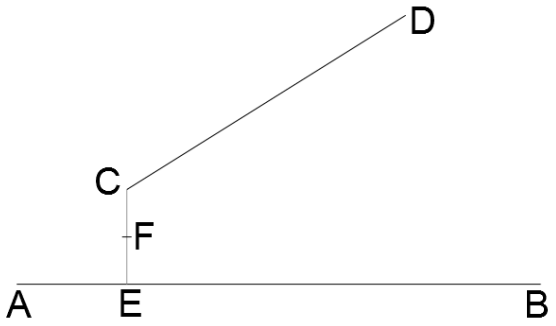


Figure 2: Showing closest middle point between two line segments.

3.2 Dual layer implicit objects

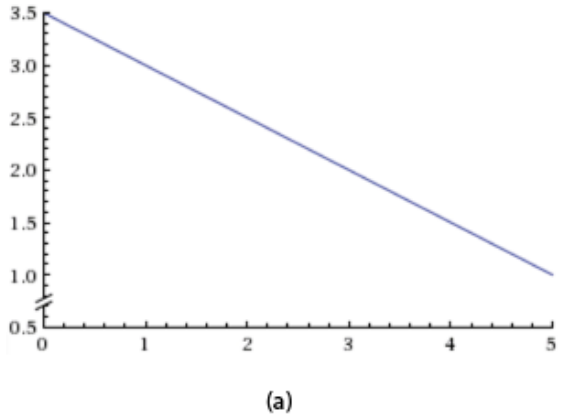
3.2.1 Object representation

The first technique that was implemented uses objects with two layers, one rigid inner layer and one deformable outer layer. The deformable layer starts where the rigid layer ends. This gives us two field functions for each object. The field functions depends on the distance, r, to the closest point on the skeleton. Figure 3 shows the profiles of the field functions for the two layers. To control the form of the objects we have three parameters. R is the scope of influence, meaning the longest distance away from the skeleton that should influence the object. r_0 is the thickness if the rigid layer, and k is the stiffness of the rigid layer. The surface of the object will be at the edge between the rigid and deformable layer.

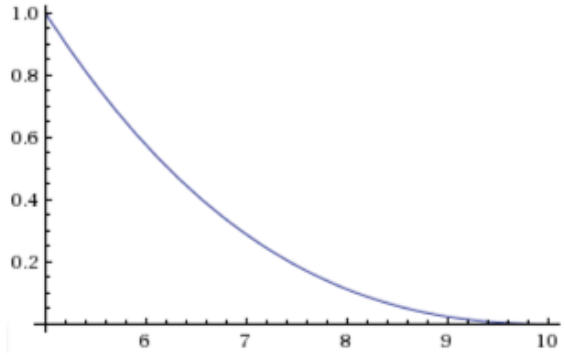
$$f(r) = -kr + kr_0 + 1, \quad 0 \leq r \leq r_0 \tag{2}$$

$$\begin{aligned} f(r) = (r-R)^2 (& \frac{r(-k(r_0-R)-2}{(r_0-R)^3} \\ & + \frac{kr_0(r_0-R) - R + 3r_0}{(r_0-R)^3}), \quad r_0 < r \leq R \end{aligned} \tag{3}$$

Both these functions evaluate to 1 when $r = r_0$, giving a smooth transition from the rigid to deformable layer. r is the distance to the skeleton. This technique originally also computed forces between the intersecting objects. Computaion of forces has not been implemented, since the focus of this project was on the visual aspect of deformations alone. This technique is described by Gascuel in [4].



(a)



(b)

Figure 3: Profile of field function for rigid layer of the object (a) and for the deformable layer of the object (b).

3.2.2 Deformation term

As stated earlier, when two objects intersect they will be deformed by adding a deformation term to the field function around the intersection area. In this approach this deformation term is a function dependent on the distance to the object that is causing the deformation. See figure 4. Along with the distance it requires the maximum compression caused by the intersection. In areas that are inside both objects we need to compress the objects so they end up just touching and not intersecting each other. The maximum compression is used to see how large the deformations should be. Little compression gives little deformation. Besides this the function takes two parameters that allow you to control the deformation. Parameter w is the maximum distance that will be used and parameter a controls the height of the deformations along with the maximum compression term.

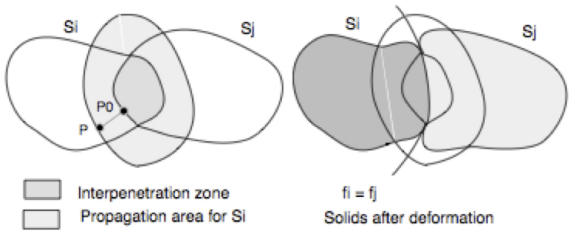


Figure 4: Distance used in deformation function. Figure by Gascuel [4].

$$deform = 4\frac{wk-4a_0}{w^3}r^3 + 4\frac{3a_0-wk}{w^2}r^2 + kr, \;\; 0 < r \leq \frac{w}{2} \qquad (4)$$
$$deform = \frac{4a_0(r-w)^2(4r-w)}{w^3}, \;\; \frac{w}{2} < r \leq w$$

$a_0 = a * maximal - compression$, r is the distance to the intersecting objects surface, and k is the stiffness value as described earlier. Figure 5 shows the profile of the deformation function. $a0$ is maximum of the function and is found when r equals $w/2$. There was a typo in the original paper that lead to some confusion. The deformation function from $w/2$ to w had a plus sign where it should be a multiplication. This lead to the two parts of the deformation function not matching at r equals $w/2$.

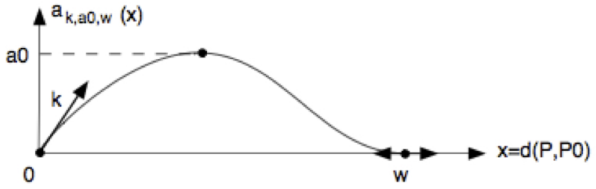


Figure 5: Profile of the deformation function. Figure by Gascuel [4].

3.3 Single layer implicit objects

3.3.1 Object representation

The second technique that was implemented uses a more simple way to represent the objects. It just uses one layer which field function is only dependent on the distance to the skeleton. The field function I have used is

$$f(r) = \frac{1}{r} - 1, \;\; 0 < r \leq 1 \qquad (5)$$

To support different widths of objects a width value k can

be added and r in the field function be replaced with r/k. Figure 6 shows the profile of the field function.

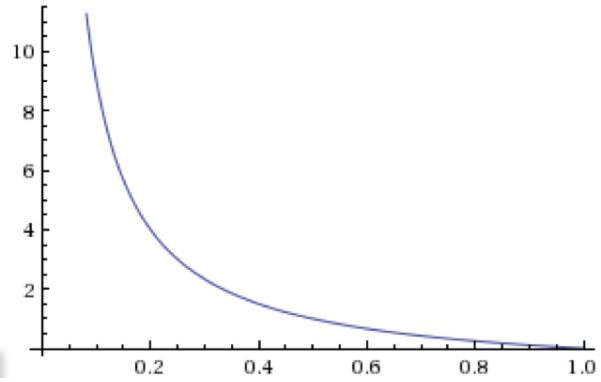


Figure 6: Profile of the field function used in the second technique.

3.3.2 Deformation term

This deformation technique was described by Angelidis et al. in [2]. This technique uses the field function of the intersecting object to deform the object. If object i and object j intersect, deformation of object i uses the field function, f_j , of object j. To control the deformation we have three parameters c_j, m, and h. c_j is the minimum value of f_j that should be part of the deformation area. m is the value of f_j where the top of the deformation should be. h is the maximum of the deformation function.

$$
\begin{aligned}
deform_i(f_j) &= (3-2\frac{f_j-c_j}{m-c_j})^2 h, \quad c_j \leq f_j \leq m \\
deform_i(f_j) &= h+((1-3h)+ \\
&\quad 2h-1)\frac{f_j-m}{iso-m})(\frac{f_j-m}{iso-m})^2, \quad m < f_j \leq iso
\end{aligned}
\tag{6}
$$

The profile of the deformation function can be seen in figure 7. In the paper by Angelidis et al. they also experimented with other deformation functions. This function is the one I found to be the best. It provides good control of the deformation without too many parameters. It should be noted however that there are other deformation functions that may be better depending on what is needed. For example, Angelidis et al. provided a function for creating ripples, in their paper[2].

3.4 Improved deformation of single layer implicit objects

A problem we found with the single layer technique was that it created too much deformation when the objects barely intersect. There was no smooth transition from no deformation to big deformation. The dual layer technique used the maximum compression term to control how big

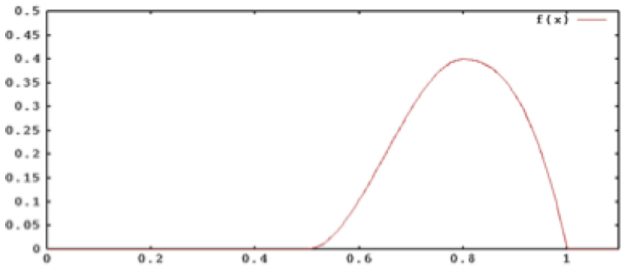


Figure 7: Profile of the deformation function used in the single layered technique. [2]

the deformation should be. By adding this to the single layer technique we were able to get a smoother transition. Instead of having a fixed maximum height of the deformation function we scale it by how much the object intersect.

4 Marching Points

Rendereing of implicit objects can be done by raycasting. Singh and Narayanan have introduced a ray casting algorithm for rendering implicit surfaces on GPU[6]. For each ray you sample the ray at a set interval looking for a root. Another way of looking at this is to have a point march down the ray and take samples, hence the name marching points. To find a root you do a sign test. You compare the current sample to the previous one and check if the sign has changed. If the sign has changed there is a root somewhere in the interval between the previous and the current sample. In figure 8 the sign will change from the sample at point B to the sample at point C. This means there is a root in the interval [B,C]. When an interval with a root is found the exact position of the root can be found using bisection. If a too large step size is used objects can be missed completely. If the object falls inbetween two sample points, the algorithm has no way of discovering the object. This makes the quality and effectiveness of the algorithm very dependent on the step size.

The rays cast using the Marching Points algorithm are independent of eachother. This allows us to cast multiple rays in parallel using the GPU. Using CUDA one thread processes one ray. For each ray the algorithm marches forward a small step at the time until it finds the interval where there is a root. When this interval is found, the root is found using bisection.

5 Implementation and Results

5.1 Framework

Our framework have been implemented using Python and CUDA. The raycasting and all evaluation of surfaces and deformations are performed on GPU using CUDA.

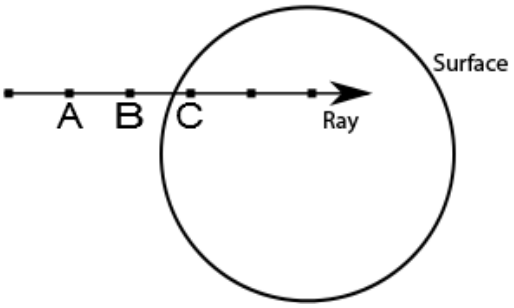


Figure 8: Illustration of Marching Points algorithm. A ray is shot towards the surface. Sampling occurs on regular intervals on the ray, like point A, B and C.

5.2 Dual layer implicit objects

The dual layer technique have been implemented as described without any alterations. To define the object we have used the parameters $k = 0.5$, $r_0 = 5$, and $R = 10$. To deform the object different parameters were tried but in the end $w = 10$ and $alpha = 0.3$ were used as default values. This technique gives very good control over the deformation and we can get deformations over a very large area if we want to.

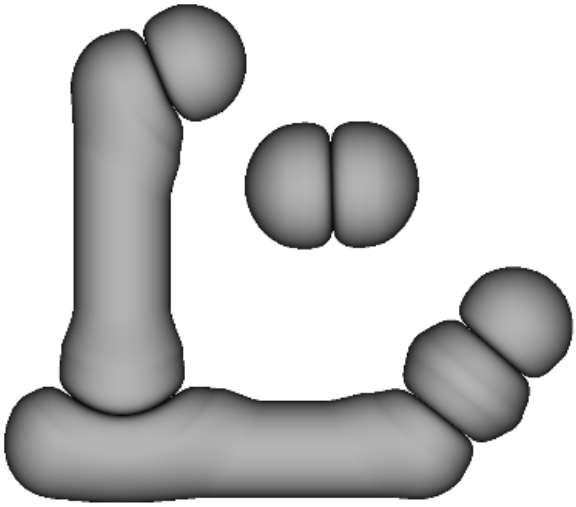


Figure 9: Rendering when using technique 1. Default parameters.

5.3 Single layer implicit objects

A problem with this technique was that the deformation when two objects barely intersect was too big. To fix this the height, parameter h, of the deformation was scaled by the maximum compression in the same way as in the dual layer technique. This gave a smoother transition from a scenario where no deformation should occur to a scenario where it should occur. This can be seen in figure 10. Figure 10 (a) does not have any scaling. This results in a too large deformation. In the figure the objects are barely intersecting and the deformation is already quite large. In figure 10 (b) scaling is applied. We downscale the height of the deformation when there is little intersection. This gives a much smaller deformation which is more fitting to how much the objects intersect.

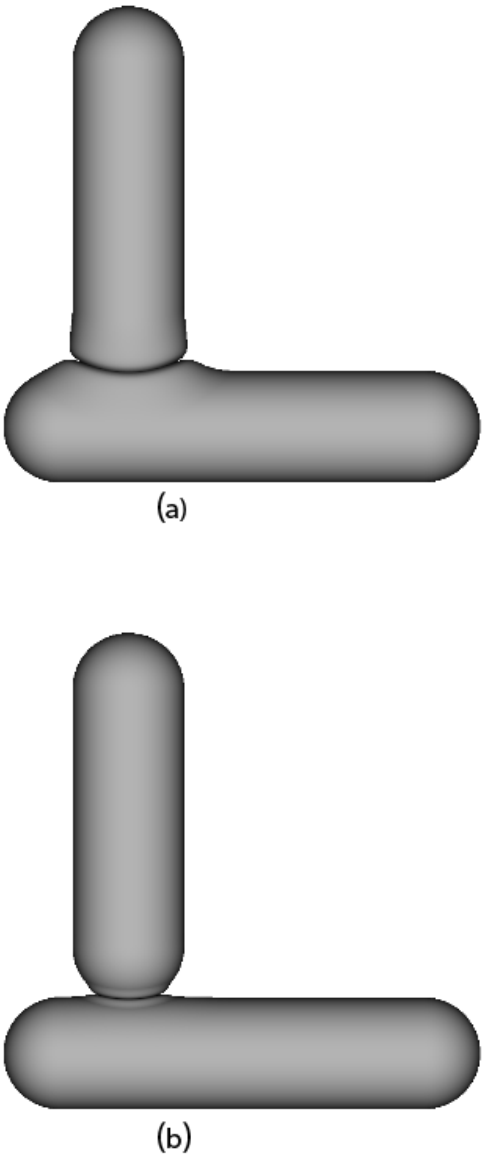


Figure 10: Resulting deformation without scaling of parameter h (a) and with scaling of parameter h (b).

The deformation term in this technique has three parameters to help control the deformation. Parameter c_j con-

trols the width of the deformation. The lower c_j the wider the deformation becomes. Parameter m controls where the top of the deformation should be, and parameter h controls how large the deformation should be. Figure 11 at the end of the paper, shows comparisons of deformations with different parameters. The figures illustrate how each of the parameters effect the deformation. As default values $c_j = 0.4$, $m = 1.3$ and $h = 0.5$ have been used. These values gave nice results, with not too much deformation and both the width and center of the deformation seemed natural. Figure 12 shows resulting rendering using this technique.

5.4 Performance

The implementation of the improved single layered technique have been tested with a 2.80GHz Intel Core i5 CPU, 8GB Memory, and a NVIDIA GeForce GTX 570 with 1280MB memory GPU. Results from testing can be seen in the table below. We have used frames per second(FPS) to measure performance.

Resolution	No. Points	No. Lines	FPS
256x256	10	0	47
256x256	5	2	23
256x256	40	0	17
256x256	50	0	16
256x256	4	3	17
512x512	10	0	24
512x512	5	2	10
512x512	40	0	7
512x512	50	0	6
512x512	4	3	7

Using this technique with only point-skeletons performs quite well. However, adding line-skeletons slows it down fast. In any case, the results are promising. The implementation can be optimized in several ways, for example by adding bounding boxes to the objects and by implementing adaptive step size in the raycasting algorithm.

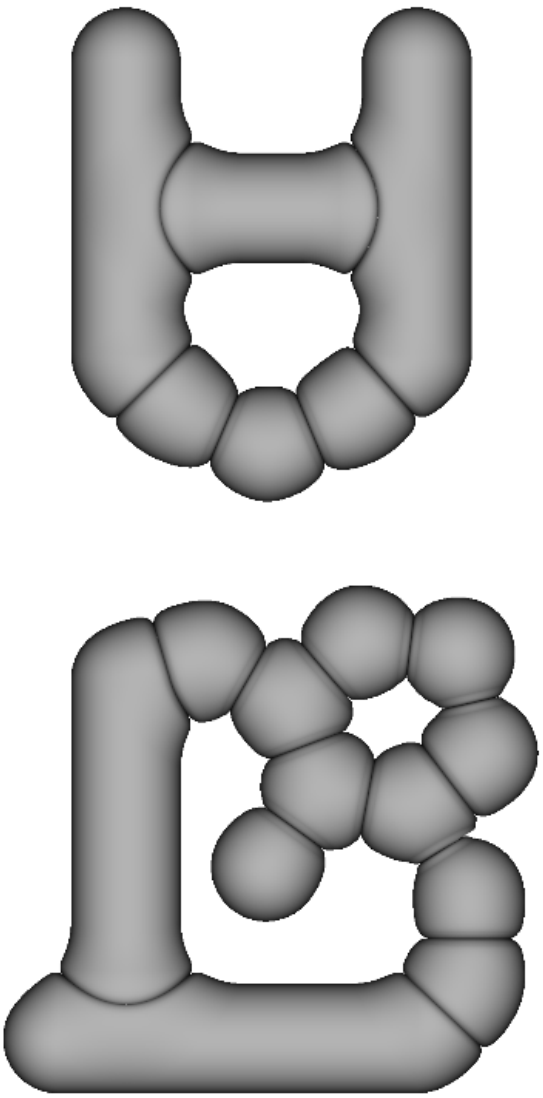


Figure 12: Resulting rendering using our technique. Default parameters.

6 Conclusions and Future Work

In concusion, we have implemented a working environment for rendering imlicit skeleton based objects and the deformation of these when they collide. The environment is interactive for both point and line skeletons. This is without any bounding boxes or any other optimizations. For future work this could be implemented to give better performance. The ray casting algorithm can be improved by implementing adaptive step size. Currently the algorithm use the same step size all the time. The step size can be varied by looking at the distance to the closest object. If the step size is set to the distance to the closest object, we avoid a lot of calculations while we are stil sure that we don't miss anything. In addition future work could look at the intersection of deformations. Currently the implementation does not check for any intersection of deformed objects, but only the original objects. This means that if the deformed sections of two objects intersect, the implementation does nothing to deform these objects further or alter the deformations in any way.

A natural extension of this work is to add physical forces to the objects. The focus of this paper was only on the visual part and adding forces was outside the scope of this work. For this paper we have used distance functions to model the objects. For future work it would be nice to have deformations working with convolution surfaces as well so it could be combined with the work from the previous project mentioned in the introduction. Another thing to consider is deformation from deformations.

As of now only the original objects are taken into account when calculating deformations.

References

[1] Alexis Angelidis. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. *seventh ACM symposium on Solid modeling*, 2002.

[2] Alexis Angelidis, Pauline Jepp, and Marie-Paule Cani. Implicit modeling with skeleton curves: Controlled blending in contact situations. *nternational Conference on Shape Modeling and Applications (SMI'02)*, pages 137–144, 2002.

[3] MP Cani. Subdivision-curve primitives: a new solution for interactive implicit modeling. *Shape Modeling and Applications, SMI*, 2001.

[4] M.P. Gascuel. An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 313–320. ACM, 1993.

[5] Agata Opalach and Marie-Paule Cani. Local deformations for animation of implicit surfaces. *13th Spring Conference on Computer*, pages 1–9, 1997.

[6] Jag Mohan Singh and P J Narayanan. Real-time ray tracing of implicit surfaces on the GPU. *IEEE transactions on visualization and computer graphics*, 16(2):261–72.

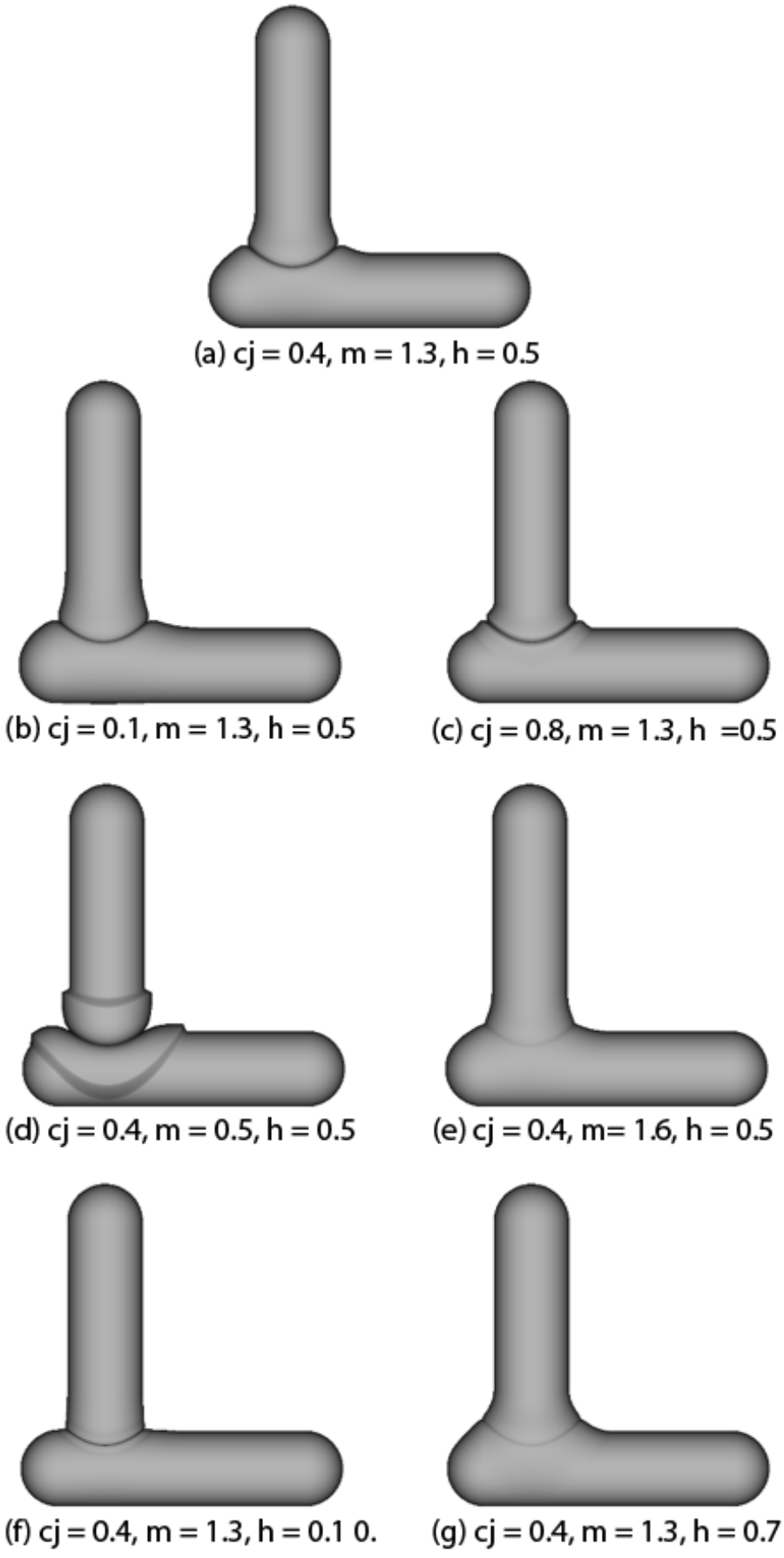


Figure 11: Deformations using our technique with different parameters

Knoocks - Ontology Visualization Plug-in for Protégé

Mgr. Adam Jurčík*
Supervised by: doc. Ing. Jiří Sochor, CSc.†

Department of Computer Graphics and Design
Faculty of Informatics Masaryk University, Brno

Abstract

When searching the World Wide Web for information we use search engines such as Google, Yahoo! or Bing. Full text search works well when finding some articles or documents, but it will not directly answer the questions as "How much something costs?" or "Where can I go for lunch?". Wanted information can be available on the Web as web pages or data (from services) but both without semantics. Ontologies are proposed by W3C to enable inclusion of semantics in web pages. This paper presents Knoocks visualization plug-in for Protégé-OWL editor. The Knoocks approach combines the traditional ontology visualization (e.g., node-link or space-filling) strategies to enhance understandability of OWL Lite ontologies. The re-implementation of Knoocks as a plug-in for Protégé-OWL extends the editor with a new ontology visualization technique and makes the Knoocks visualization publicly available for users. The Knoocks plug-in is good at visualizing of non-trivial ontologies.

1 Introduction

Automated processing of data contained in web pages is impossible without having knowledge of its semantics. In computer science, ontologies represent knowledge of a domain as a set of concepts and relationships between them. Building an OWL (*Web Ontology Language*) ontology and marking up a web page will enable web page's content to be understood by machines [1]. To build an OWL ontology, Protégé-OWL editor[1] (publicly available as freeware) can be used. The editor allows people to manipulate OWL entities such as classes, object/data properties and individuals. The ontology is itself a set of axioms that define entities and relationships between them, the axioms are more easily understood when they are visualized.

The Protégé-OWL editor has basic tree views that show hierarchical relationships between entities. Currently available visualization plug-ins for Protégé-OWL, e.g., OntoGraf, NavigOWL or SOVA visualize ontologies as interlinked nodes. The node-link visualization approach [3] draws classes and individuals as nodes, relationships between entities (hierarchical or property) are drawn as links. On the contrary, space-filling visualizations group entities together by classes or by properties. Groups are drawn as areas that can be included by other areas that represent more general concepts.

Node-link visualizations are optimal in presenting parts of ontologies in detail, while when visualizing overview of an ontology the resulting visualization can become confusing, hard to understand. Finally, it is up to the user to lay out nodes to create understandable presentation of chosen concept. Space-filling visualizations give users very clear idea of hierarchical relationships between entities, i.e. subclasses, or general concepts in ontologies – subdivision of an ontology into smaller logically connected parts. Lack of links in pure space-filling visualizations (see CropCircles[2] [11]) does not allow to express dependencies (typically object property instances) across entities. Therefore, combination of both node-link and space-filling approaches are examined to create more universal visualizations (see Jambalaya[3] [10]).

A Protégé-OWL visualization plug-in implementing Knoocks visualization approach was developed as a part of master's thesis at our department. The Knoocks (stands for Knowledge Blocks) approach was designed by Kriglstein and Motschnig-Pitrik as a combined visualization approach [7]. This paper will present only key concepts of suggested visualization approach. The plug-in was implemented in Java as an OSGi (*Open Services Gateway initiative*) bundle for Protégé framework[4] and uses OWL API and Protégé-OWL API to introspect the visualized ontology. Architecture of the plug-in and some chosen implementation details are presented by this paper. Also an example of usage is included which demonstrates how to benefit from Knoocks when getting familiar with an unknown ontology.

2 Related work

In 2008 Kriglstein showed advantages of representing university curricula as ontologies. The curricula that were rolled out at University of Vienna in 2006 were organized into modules forming a clear structure. The structure of

*xjurc@fi.muni.cz
†sochor@fi.muni.cz

[1] http://protege.stanford.edu/overview/protege-owl.html

[2] http://www.mindswap.org/2005/cropcircles/

[3] http://www.thechiselgroup.org/jambalaya

[4] http://protege.stanford.edu/

a curriculum can be represented by hierarchy of classes and contained courses are in ontologies included as individuals [4]. When representing curricula as ontology it is easy to express (using object properties) arbitrary relationships between courses, modules and other objects. Visualization of curriculum's ontology helps teachers and students to understand it, e.g., it is easier to see dependencies among courses. Ontologies also enable to share or analyze the knowledge contained in a curriculum. In her paper, Kriglstein compared several ontology visualization techniques in the context of curricula visualization and concluded that most requirements were met by Jambalaya.

However, Kriglstein and Motschnig-Pitrik identified disadvantages in existing ontology visualizations and proposed Knoocks as a new visualization approach [7]. Newly proposed approach targeted on three main points.

- **Visualization of ontology overview** – an overview [9] of structure should help user to understand and navigate large ontologies that contain multiple subhierarchies of classes.
- **Visualization of classes** – every class should be easily seen and also names of classes should be visible to distinguish among them.
- **Visualization of individuals** – individuals represent concrete objects and therefore it is important to show them. Also names and classification of individuals should be visualized to make understanding of ontologies easier.

Knoocks uses space-filling visualization to show hierarchy of classes as a block. Each class is represented as a rectangle contained in the block. Furthermore, classes contain individuals to emphasize their classification. First version of Knoocks was implemented in Java and was able to visualize only one class hierarchy. Therefore the requirement of ontology's overview visualization was not met.

Further development of Knoocks was based on user requirement analysis that was conducted by University of Vienna and focused on ontology experts and semi-experts [5]. Interviews and online survey were organized to gather users' requirements on ontology visualizations. The analysis showed that users do not clearly prefer one visualization tool over another. On the other hand, users' expectations about general ontology visualization were similar in few points. Users stated that a good visualization should have both overview and detail views, browsing and updating should be possible and relationships between entities (subclass or object properties) should be visible.

Ontology overview and data/object properties visualization was provided by a new prototype of Knoocks [8]. The prototype was written using C# and OpenGL as a standalone application. This version of Knoocks added switching between overview, which shows an ontology as a set of blocks, and detail view, which shows more details about a block. In overview, object properties between individuals are visualized as edges (node-link approach is used) between their classes. Details about object properties, e.g., which individual is related to which, are available in detail view of a block [6]. Also data properties of an individual can be made visible in detail view. Layout of visualized individual details was chosen after evaluation of its usability by users. Knoocks prototype was further extended to include searching and filtering of entities, and also navigation among detail views based on class/individual selection was added. The Knoocks prototype was taken as a specification for Knoocks plug-in implementation for Protégé-OWL editor. Therefore, Knoocks visualization concepts that were implemented by the plug-in will be described in more details in the next section.

3 Features of Knoocks plug-in

Knoocks plug-in is available in Protégé-OWL as a tab plug-in. Tab plug-ins can be included or excluded from editor's view – configuration is done my using *Window/Tabs* menu. Newly installed tab plug-in is hidden by default.

The Knoocks tab is vertically split into two panels (see Figure 1), where left panel contains user controls for ontology searching and filtering, and right panel contains interactive visualization of an active ontology – the editor allows to manage more ontologies at a time.

The Knoocks prototype provides also a preview view that is in our plug-in missing yet.

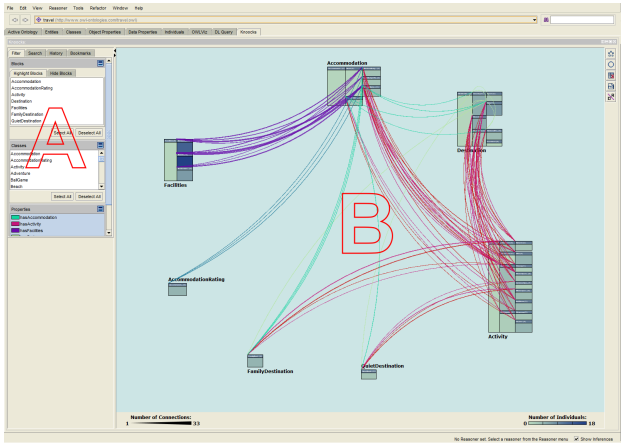
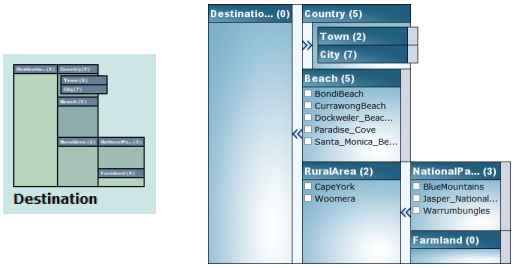


Figure 1: Knoocks tab plug-in in Protégé-OWL. The plug-in's view contains the control panel (A) and the visualization slot (B).

3.1 Knowledge block

Arrangement of class hierarchies into blocks is a key concept of Knoocks approach. Each class that is a direct subclass of the `OWL:Thing` class becomes a root of a block. The block is named after its root class. A block is laid out (see Figure 2) in a way such that subclasses are placed on the right side of their parents. Classes are drawn

as rectangles with captions that show class names (visible only in detail view). Individuals are displayed within their classes. Therefore, the amount of space occupied by a block depends on width of its hierarchy branching and also on count of individuals contained in classes. Classes in a block can be folded to occupy less space in the visualization.



(a) Overview (b) Detail view

Figure 2: Example of a block.

3.2 Overview

Laying out all blocks from an ontology gives users a good view on its structure. In overview, blocks are drawn with less detail, e.g., class names are provided rather as tooltips and individual names are not drawn in class rectangles. At first, blocks are automatically arranged into circular shape (see Figure 3), but they can be moved, so that users can control the visualization themselves. Blocks can be rearranged to circular layout when needed. Bodies of classes are drawn with different shades to visualize cardinality of classes. The darker the class is the more individuals it contains. Navigation to detail view of a block is done by double-clicking it.

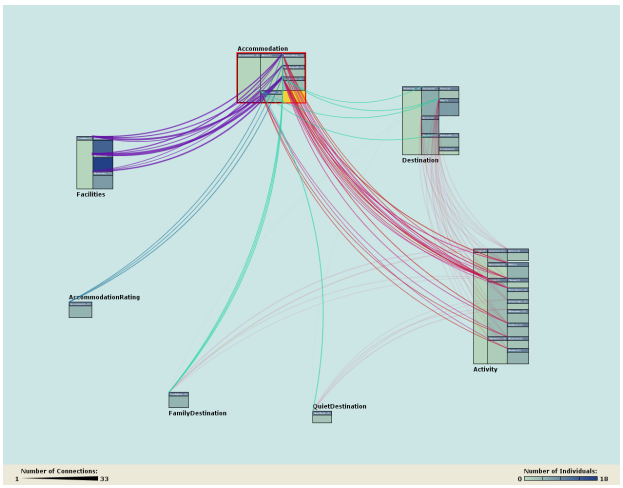


Figure 3: Example of overview of an ontology, blocks were laid out automatically.

Colored connections are drawn to visualize relationships between blocks/classes. Different colors are chosen to distinguish among object properties. Edges are not drawn directly between individuals (where instances of object properties are defined), they are grouped to connections by their class paths (domain and range) instead. Thickness of a connection depends on count of object property instances that were grouped, so that users can quickly infer how much are the two classes interconnected. Visualization of object properties is interactive, a connection can be clicked to display a table which lists concrete relations that were grouped. Classes and individuals in the connection table are also clickable and allow navigating to them, i.e., they are shown in detail view when clicked.

When a block is selected by clicking on it, connections that are not related to selected block are faded out, so that block's connections are emphasized.

Overview provides also useful statistics about ontology visualization. The statistics are located at the bottom panel in the view and shows maximum connection and class cardinalities among the ontology.

3.3 Detail view

Detail view allows users to focus on a block that is of their interest. Class names are visible in class captions and listings of individuals are shown for non-empty classes. A class can contain many individuals, and therefore paging of listing is used – the listing could become very long and would require classes to be too high. A bar button with arrow is added between each parent class and its children to control folding of classes. Class captions and individuals are clickable to allow users to get details about them.

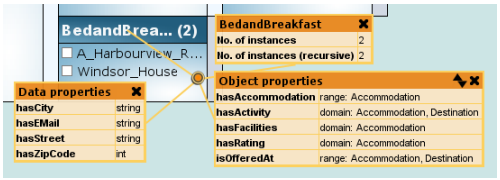


Figure 4: Details of `BedandBreakfast` class.

Class details include statistics about its individuals, data types of related data properties and domains or ranges of related object properties. All three types of information are shown in separate pop-up tables (see Figure 4) that can be moved (independently or together). When an individual is clicked, values of its data properties and its relations to other individuals are visualized. Values of data properties are shown in a pop-up table – in the same way as data types of a class. Relations to other individuals are grouped by their class paths and shown as listings – one for each different class path (see Figure 5). The table and the listings can be moved (same as class details), closing of shown details is done by closing data property values table. In addition, classes and individuals that are displayed in a relation listing are clickable, so that they allow navigation in the visualization.

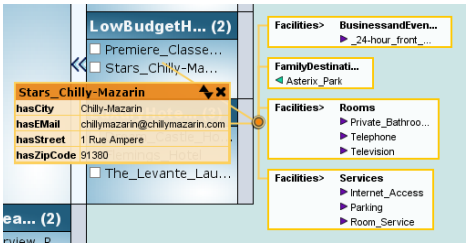


Figure 5: Details of `Stars_Chilly-Mazarin` individual.

3.4 Filtering and Search

Knoocks visualization usability is improved by filtering of ontology entities. In overview, visibility of blocks and connections can be controlled. A block can be hidden by selecting a row with its name in *Hide Blocks* list. Visibility of connections depends on visibility of object properties and is set in *Properties* list. Blocks and classes can be highlighted to emphasize them in the visualization. The highlight is visible in both overview and detail view. For users' convenience, visibility of object properties and block/class highlights can be controlled from context menus too.

The plug-in allows for basic search in the visualized ontology. *Search* panel is used to retrieve matching classes and individuals. Classes are typeset bold in the results list. Clicking on a class or an individual that were matched navigates in the ontology to it and shows it in detail view. Almost every navigation (by clicking) to a class or an individual is recorded and shown in *History* panel. The *History* panel allows users to go back where they came from, this is often useful when inspecting relationships in an ontology. Jumps into history are not recorded again. Furthermore, to make navigation in big ontologies easier, individuals can be bookmarked. Bookmarked individuals can be found in *Bookmarks* panel. When a bookmark is clicked, the view is navigated to particular individual.

Highlights of blocks and classes are visible in all lists that show their names.

4 Implementation

Protégé is modular Java framework for knowledge-base management. The framework is based on OSGi service platform specification[5]. Protégé-OWL editor for OWL ontologies is implemented as a module (bundle in OSGi) which uses and extends functionality of core Protégé. Two versions of Protégé are maintained, because they provide different features. Protégé 4 supports OWL 2.0[6], therefore it was chosen as a target for Knoocks plug-in. The plug-in's compatibility was tested against Protégé 4.1.

[5] http://www.osgi.org/Specifications/HomePage
[6] http://www.w3.org/TR/owl2-overview/

4.1 Architecture

Knoocks visualization plug-in is implemented in Java 6 SE using mainly Swing and Java 2D. The plug-in is distributed as an OSGi bundle and depends on Protégé, Protégé-OWL[7] and OWL API[8] bundles. Protégé-OWL classes are more convenient to use for ontology inspection than using OWL API directly. Furthermore, the plug-in utilizes Commons-Collections with Generics[9] library which is a Java 1.5 port of popular Commons Collections project. Architecture of the plug-in is shown in Figure 6 using UML (*Unified Modeling Language*).

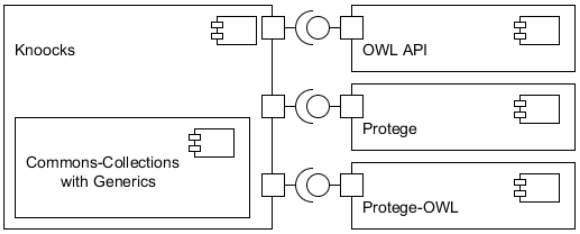


Figure 6: Architecture of Knoocks plug-in (*Component Diagram*, UML).

4.2 Design

Knoocks ontology visualization is interactive and driven by user events. Therefore, the plug-in's code was designed using the MVC (*Model-View-Controller*) pattern[10]. The UI (*User Interface*) part of plug-in's code is tightly coupled with Swing which implements its own modified MVC. In the process of development, the IoC (*Inversion of Control*) principle[11] was identified to be beneficial to use. It was decided that the plug-in is too small to utilize an IoC framework, so that a simple mechanism of handling dependcies was designed and implemented.

4.2.1 Model

The OWL API already models entities of an OWL ontology, but there were reasons to design an overlaying model. The Knoocks approach contains blocks and connections that are not included in the OWL API. Blocks are hierarchies of classes that have to be inspected and remembered, because users are allowed to fold/unfold classes which requires a knowledge of class' hierarchy. Connections group relations among instances by their class path. It is beneficial to extend the new model with these specialized entities. All other entities from OWL API, namely class, individual, data property and object property, have

[7] http://protege.stanford.edu/plugins/owl/api/
[8] http://owlapi.sourceforge.net/
[9] https://github.com/megamattron/collections-generic
[10] http://ootips.org/mvc-pattern.html
[11] http://www.oodesign.com/dependency-inversion-principle.html

their counterparts in plug-in's model. Entities from OWL API model do not allow to get their names directly (using entity's method), OWL entity rendering must be used instead. Knoocks uses same name rendering throughout the visualization, therefore it was easy to include methods that provide entity names in our model. In addition, a few properties regarding visualization were added to some entities, e.g., the new class entitiy has a property named *highlighted* which tells the view whether user wants that class to be rendered as highlighted or not.

4.2.2 Controller

Controllers are classes where logic is contained in MVC applications. In Knoocks plug-in, controllers take care of switching view between overview and detail view, provide searching and navigation and modify the model that is used by views. Every action that is initiated by a user through the view part of Knoocks plug-in is executed by some controller. Controllers are cooperating together to improve reuse of the code and distribution of logic among appropriate classes. Each block in the plug-in has its own controller object which is instanced from appropriate class.

4.2.3 View

Knoocks' views are implemented using Swing and Java 2D for custom drawing. Each plug-in's view is a Swing component. The top level view contains control view as a subview and defines a slot that allows switching between overview and detail view (see Figure 1). The definition of subviews is supported by Swing's laying out of child components. Four new UI components were developed for visualization of blocks. Two components represent whole blocks (in detail view and in overview) while another two components take care of class rendering in appropriate views. Custom layout manager (feature of Swing UI framework) was developed for supporting of automatic layout of classes in a block. The layout manager takes into account folded or unfolded state of clasess in a block.

5 Usage example

Google and Yahoo SearchMonkey support crawling of e-commerce data enriched with concepts from GoodRelations vocabulary [2]. The GoodRelations semantic vocabulary (in OWL 2.0) allows to add business information into web pages. Semantics is marked up using RDFa (*Resource Description Framework in attributes*) or Microformat specifications. Semantically enriched data is machine understandable, therefore it is not necessary to implement specialized e-commerce APIs (such as Google's Content API) to make offered products reachable by search engines. Although the GoodRelations ontology has quite shallow hiearchy it is an example of OWL ontology that is widely used.

Protégé with Knoocks plug-in can be used to quickly become familiar with the ontology. The overview (shown in Figure 7) gives view on general structure of a part of the ontology. Chosen part visualizes all classes that are available for product description. Block arrangement of class hierarchies points out existence of subclasses – specializations of basic concepts. Classes painted with different shades make distribution of ontology's individuals visible at first glance. Definitions of object properties are rendered instead of connections when the view is set to draw arrows between domains and ranges. Rendering of properties is controlled by a button at Knoocks' toolbar (see Figure 8).

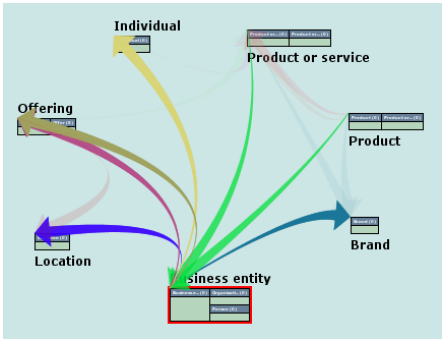


Figure 7: Object properties related to `Business entity` class.

Figure 8: Toolbar.

Detail view can be used to inspect a block. Figure 9 shows detail view visualization of `Payment method` class from the GoodRelations ontology. Names of individuals are drawn in class bodies regarding their classification. Information about data and object properties related to a class will be shown when clicking its header. Furthermore, object property assertions between individuals from `Day of week` class are not visualized in overview (as connections) because the domain and the range of both `has previous` and `has next` properties are same. All assertions related to an individual are shown when the individual is clicked.

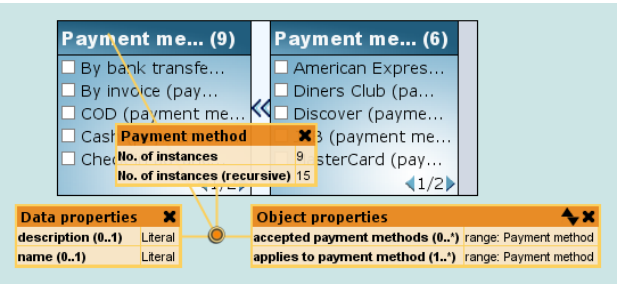


Figure 9: Visualization of `Payment method` block. The block is shown in detail view.

6 Distribution

Plug-ins for Protégé are distributed as OSGi bundles, i.e., JAR archives that provide OSGi metadata. A plug-in is installed by copying it into *plugins* directory that is located at root of Protégé's distribution. Plug-ins are automatically loaded when Protégé-OWL is started.

```
ProtegeInstallDir
└─plugins
  └─cz.muni.fi.knoocks.jar
```

At the time of writing, Knoocks plug-in is in process of publishing at Protégé's *Plugin Library*[12]. The library is considered as a standard distribution platform for Protégé plug-ins. Our plug-in will be labeled with *Visualization* category and annotated with compatibility and dependency information. Each plug-in has its wiki page that can be used by authors to provide additional information such as basic *User's Guide* or link to repository with source code.

7 Conclusion

In this paper, we described Knoocks visualization plug-in for Protégé-OWL editor. We commented briefly on Knoocks visualization approach which combines both node-link and space-filling approaches to visualize general structure of an ontology and also its details, i.e., individuals with linked values and object property assertions. By included example, we showed that our plug-in is really helpful in visualizing new ontologies that a user wants to become familiar with.
Main part of the plug-in was implemented in my master's thesis. Currently, the resulting plug-in is being made publicly available at Protégé's plug-in library, so that wide community of Protégé-OWL users will get a new visualization tool. It is expected that the use of the plug-in will lead to new requirements on Knoocks visualization approach.

References

[1] G. Antoniou and F.V. Harmelen. *A semantic Web primer*. Cooperative information systems. MIT Press, 2004.

[2] M. Hepp. GoodRelations : An Ontology for Describing Products and Services Offers on the Web. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008)*, volume 5268, pages 332–347, 2008.

[3] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods—a survey. *ACM Computing Surveys*, 39(4):10–es, November 2007.

[4] S. Kriglstein. Analysis of Ontology Visualization Techniques for Modular Curricula. In *Proceedings of 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for Education and Work*, pages 299–312, 2008.

[5] S. Kriglstein. User Requirements Analysis on Ontology Visualization. In *2009 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 694–699. IEEE Comput. Soc. Press, March 2009.

[6] S. Kriglstein. OWL Ontology Visualization: Graphical Representations of Properties on the Instance Level. In *2010 14th International Conference Information Visualisation*, pages 92–97. IEEE Comput. Soc. Press, 2010.

[7] S. Kriglstein and R. Motschnig-Pitrik. Knoocks: New Visualization Approach for Ontologies. In *Proceedings of 12th International Conference Information Visualisation*, pages 163–168. IEEE Comput. Soc. Press, 2008.

[8] S. Kriglstein and G. Wallner. Knoocks - A Visualization Approach for OWL Lite Ontologies. *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 950–955, February 2010.

[9] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *Visual Languages, 1996. Proceedings., IEEE*, pages 336–343, 1996.

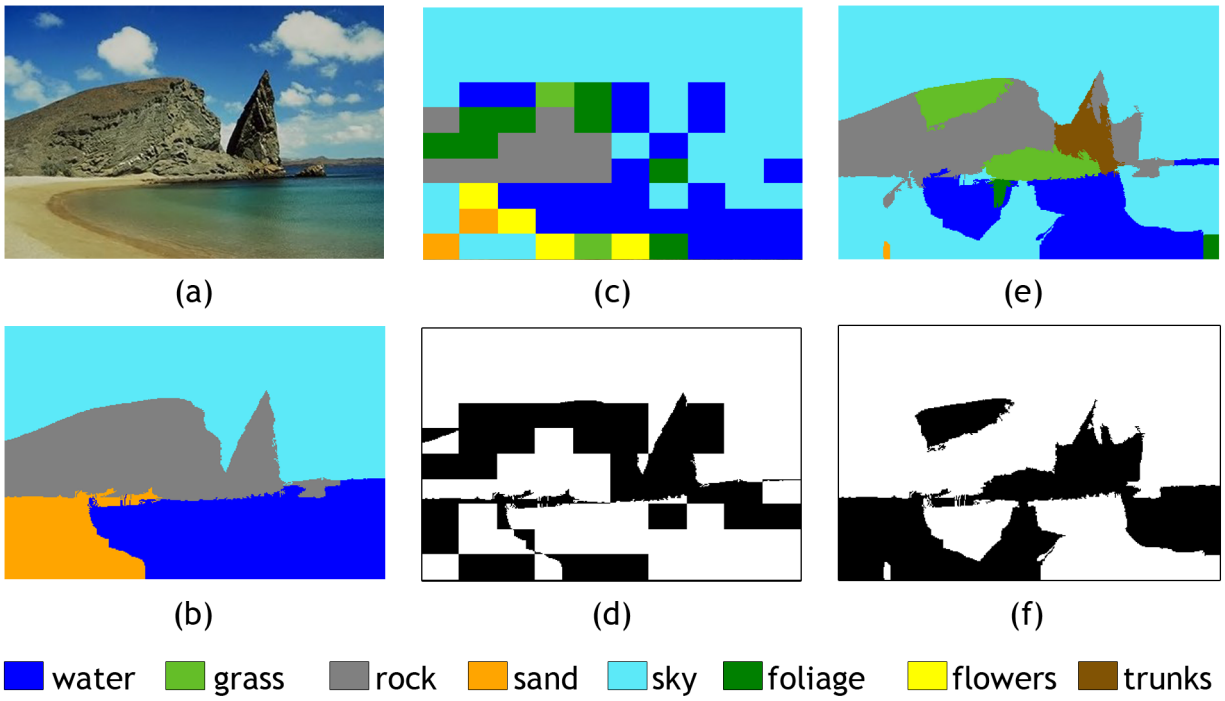
[10] MA. Storey, N.F. Noy, M. Musen, C. Best, R. Fergerson, and N. Ernst. Jambalaya: An Interactive Environment for Exploring Ontologies. In *IUI '02 Proceedings of the 7th international conference on Intelligent user interfaces*, page 239, 2002.

[11] T. D. Wang and B. Parsia. CropCircles : Topology Sensitive Visualization of OWL Class Hierarchies. In *Proceedings of the 5th International Semantic Web Conference ISWC 2006*, pages 695–708, 2006.

[12] http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library

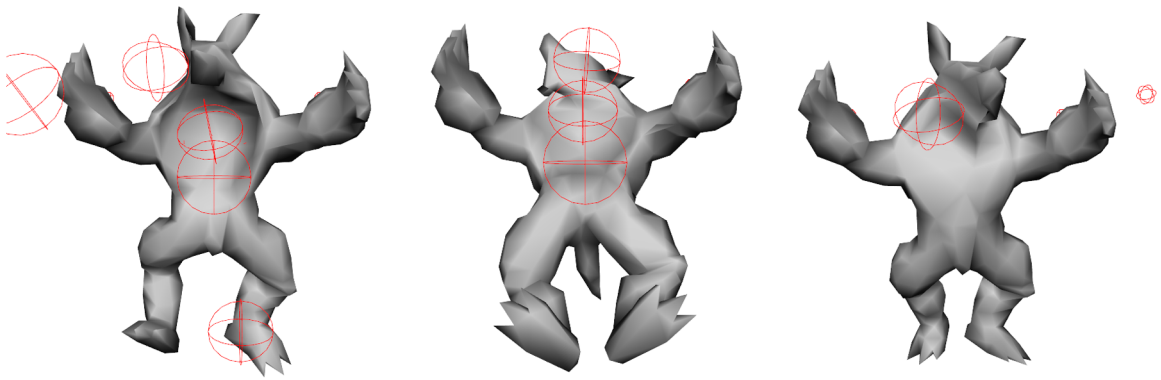
Color Plates

Kristína Lidayová
Semantic Categorization and Retrieval of Natural Scene Images
pp. 57–64



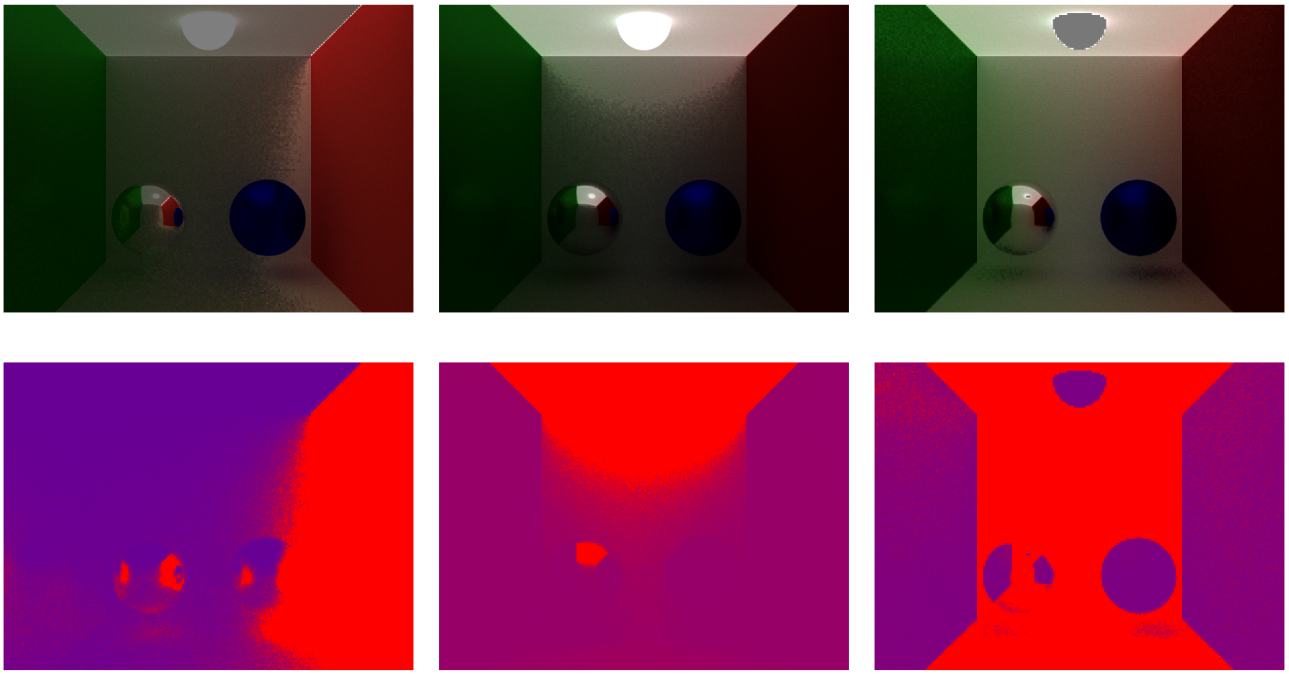
Local semantic concept classification. (a) Original image. (b) Ground truth. (c+d) Result of initial method and equality map. (e+f) Result of our proposed method and equality map.

Karolina Lubiszewska
Workflow for real-time simulation of deformable objects
pp. 83–90



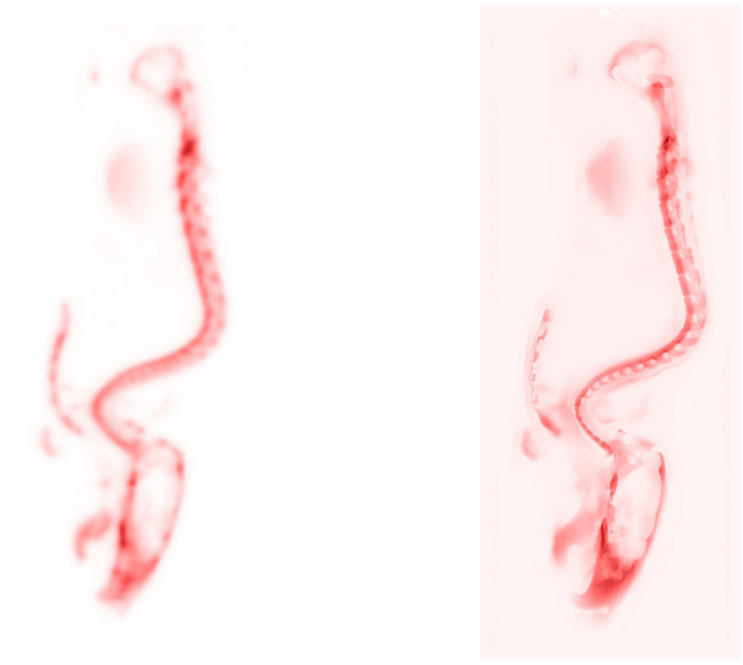
Frames depicting throwing balls at a soft body rigged with a rigid skeleton. The shape is retained much more firmly and the character does not expose unnatural behaviour.

Bernhard Kerbl
Priority-Based Task Management in a GPGPU Megakernel
pp. 99–106



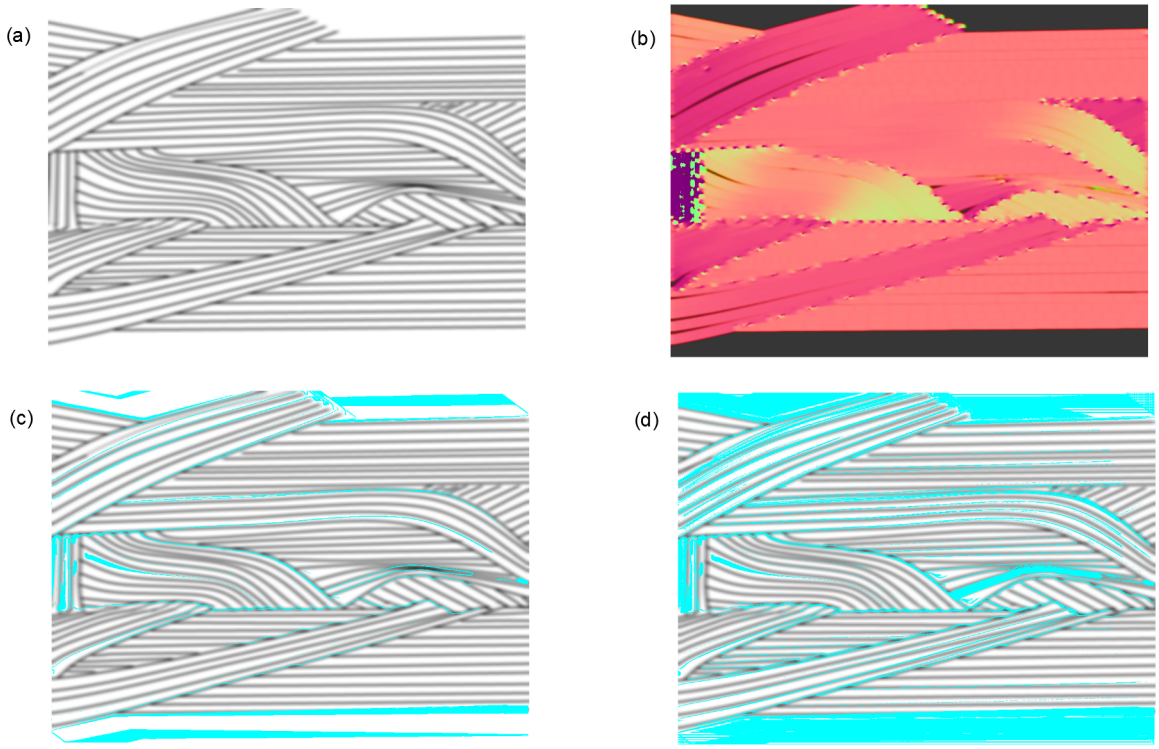
Left: Task management achieves early prioritization of primarily red areas. Middle: Intermediate stage for rendering using intensity as priority. Right: Difference of color values is used to render areas with low convergence preferably.

Zsolt Márta
Partial Volume Effect Correction on the GPU
pp. 177–184



Left: Initial PET measurement of a mouse ($136 \times 132 \times 330$). Right: Output of our algorithm ($435 \times 422 \times 1056$).

Kari Ringdal
Flow-based Segmentation of Seismic Data
pp. 185–192



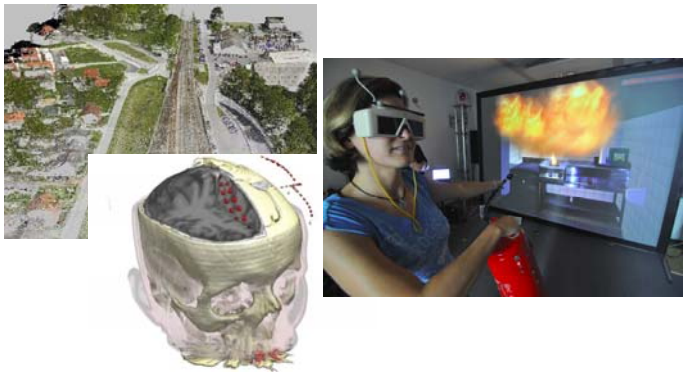
Test set 4. a) A synthetic seismic slice containing many sedimentary units. The image was copied and stacked to create a volume. b) A rendering of the RGBA-vectors that constitute the extracted flow field. c) Output of the border detection algorithm with the distance threshold set to 16 units. d) The distance threshold is set to 4 units.

Philip Voglreiter
Stochastic Particle-Based Volume Rendering
pp. 193–200

Figures show different volumes rendered with the Particle Based method. Left figure shows a simulated dataset with 6 million subpixels. The figure in the middle and the right show the stanford dragon volume, rendered with 60 million particles and subpixel level 6 and 3 respectively.

Sponsors of CESCG 2012

zentrum für
virtual reality und visualisierung
forschungs-gmbh



The VRVis Research Center

The VRVis Research Center is a joint venture in research and development for virtual reality and visualization. VRVis was founded in 2000 as part of the Austrian Kplus program to bridge the gap between academic research and commercial development as well as to supply the necessary transfer of knowledge between the academic community and industry. VRVis is now a COMET K1 center.
This mission is mirrored in a variety of academic and industrial partners. The research center is currently conducted by five academic institutes and numerous industrial partners. Leading-edge innovations and down-to-earth business style characterizes VRVis as a valued partner for high-level research.
The company's headquarter is located in Vienna, Austria. Today, around 50 researchers together with about 20 students do high-level applied and basic research in five different areas.

The Team of VRVis

VRVis consists of internationally experienced researchers in the areas of visualization, rendering and visual analysis. Their outstanding experience and knowledge in these topics qualify them for the innovative research they are performing. The research areas are headed by key researchers who manage these areas, define goals and projects for this area, and conduct the defined research together with their staff. All members of the research team are young researchers, whose creativity and ingenuity is the key to the success. VRVis is always looking for young, talented, and motivated researches in the fields of research to extend its research work or to support partner companies.

Research Program of the VRVis

The scientific research program consists of three research areas in which thematically matching research projects are conducted. Each research area realizes application projects on the one hand and basic research for these application projects on the other hand.

- Research Area Visualization
- Research Area Rendering
- Research Area Visual Analysis

Working at VRVis

VRVis is always looking for students, junior and senior researchers who want to join the VRVis team. VRVis is offering internships, diploma theses, PhD theses and regular positions. For more information please refer to the additional information listed below.

Some Partners of VRVis

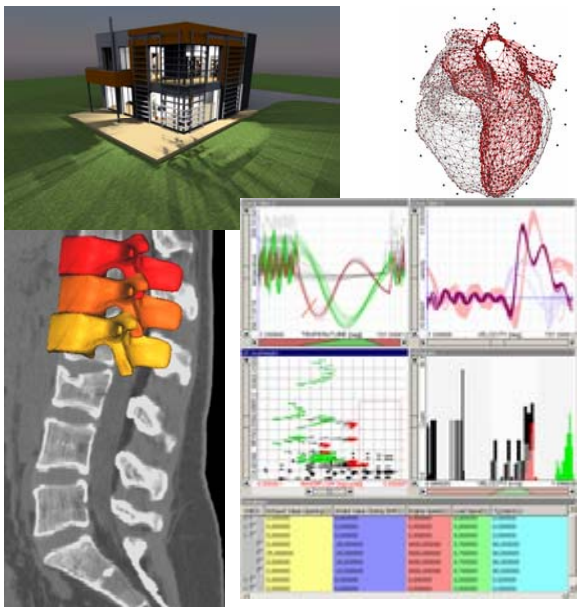
Scientific Partners of VRVis:

- Institute of Computer Graphics and Algorithms, Vienna University of Technology
- Institute of Computer Graphics and Vision, Graz University of Technology

Industrial Partners of VRVis:

- AVL List GmbH, Graz
- Agfa Healthcare, Wien
- Eybl Development GmbH, Krems
- Geodata Ziviltechniker GmbH, Leoben
- Imagination Computer Services, Wien
- ÖBB Infrastruktur Bau AG, Wien

Currently, VRVis is again extending its industrial base with new partners from several new fields.

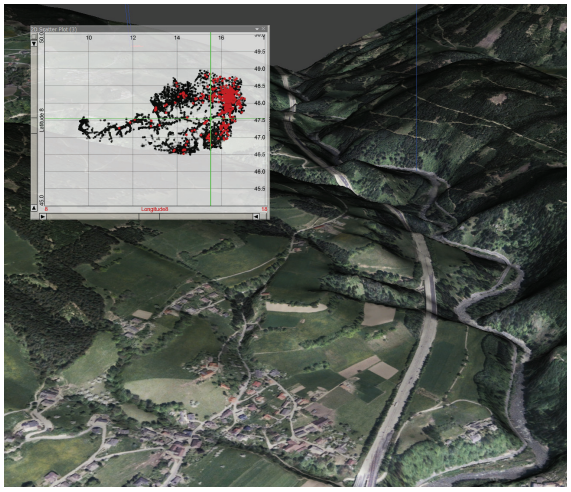


Additional Information and Contact

For detailed information about the research program, current projects and job opportunities please visit our web pages at http://www.VRVis.at/.
If you need additional information or search for job opportunities in VR or visualization, please feel free to contact Prof. Werner Purgathofer (VRVis Scientific Director) at Purgathofer@VRVis.at or +43(1)20501/30155; Donau-City-Straße 1, A-1220 Wien.

Open PhD Position in Visual Analytics at VRVis, Vienna, Austria

The Visual Analysis group at VRVis Research Center (www.vrvis.at) Vienna, Austria has an open full-time PhD research position in scope of the research project **VISAR – Visual Analytics and Rendering.**



Conceptual image illustrating navigation in 3D space by selection in a scatter plot.

The overall goal of VISAR is to achieve an unprecedented level of integration between realistic real-time rendering of 3D geometry and techniques from the field of information visualization and visual analytics. Potential application areas include the construction and inspection of large infrastructure, traffic visualization, and archaeology. The main focus of this PhD position will be to enrich a multivariate analysis of many 3D objects (e.g., cars, traffic signs, or archeological artifacts) by appearance- and visibility-related information, and to dynamically update visualization- and interaction-components at changes of either the scene or the viewpoint of the camera. The realization will be based on extending and integrating powerful software frameworks of VRVis.

We are looking for a highly motivated young scientist who intends to do a PhD in scope of this project. The starting date for the position is end of summer 2012 and the duration is 36 months. The scientific supervision will be by Harald Piringer, VRVis, in collaboration with Prof. Eduard Gröller, Institute of Computer Graphics and Algorithms (ICGA), Vienna University of Technology (www.cg.tuwien.ac.at). The minimum gross salary is 2.300€ per month (14 times per year).

To apply for the position, you should fulfill the following requirements:

- Master in computer science (or similar)
- You intend to do PhD research and intend to publish high-impact scientific papers
- Good programming skills (ideally: C++) and knowledge in software engineering
- Basic knowledge in the fields of information visualization and real-time rendering
- Very good English in speaking and writing; German is appreciated, but not required
- Creativity, common sense, and social integration

If you are interested in the position, please apply via email until **31st of May 2012** to Harald Piringer (hp@vrvis.at). After this date, further applications will be considered until the position is filled. Women are especially encouraged to apply. Your application should include:

- Curriculum vitae
- Publication list, including talks, master thesis, projects that are online available, etc.
- A short personal statement, why you in fact apply for this position
- Names of references

v r vis

FWF
Der Wissenschaftsfonds.

www.sfera.sk

PRIESTOR pre váš úspech

UŽ 20 ROKOV

sféra®

Grafické informačné systémy

Rok 2012 je dvadsiatym prvým rokom pôsobenia spoločnosti na slovenskom, českom, maďarskom a rakúskom trhu. Systém manažérstva kvality uplatňovaný v spoločnosti sféra, a.s., už od roku 2003 je aj dnes v súlade s požiadavkami medzinárodnej normy ISO 9001:2008. Naďalej zostáva v platnosti motto spojené s vyhlásenou politikou kvality "Čo včera stačilo, je dnes už málo".

Spokojnosť zákazníkov a kvalita našich produktov sú hlavnými kritériami pri plánovaní ďalšieho smerovania našej spoločnosti.

Sme pripravení plniť náročné požiadavky trhu z hľadiska aplikovania moderných technológií, zabezpečenia spoľahlivosti a bezpečnosti informačných systémov, správy a ochrany údajov a pridanej hodnoty pre stovky spokojných používateľov našich riešení.

Používanie progresívnych technológií zabezpečujeme kvalifikovaným tímom, spoluprácou a partnerskými vzťahmi s poprednými svetovými dodávateľmi informačných technológií - Microsoft, Bentley, Oracle, Hewlett-Packard, Cisco a Google. Samozrejmosťou pri nasadzovaní informačných systémov je implementácia webových a mobilných technológií v sieťovej prevádzke aj na tabletoch a PDA zariadeniach, čo umožňuje sprístupnenie profesných informácií na každej pracovnej stanici zákazníka, ako aj v teréne, bez vynaloženia neprimeraných nákladov.

Naša spoločnosť je tu pre Vás a ponúka profesionálne zdroje a spoluprácu počas celého životného cyklu informačných systémov u Vás. Preto je sféra "priestor pre Váš úspech®".

sféra, a.s. • XMatik®.NET • XMpad®.NET • XMtrade® • Bentley • ENERGOFÓRUM® • Semináre • MicroGRAP/RS® • MicroKataster • ArchiFM • Madam