Proceedings of the

14th Central European Seminar on Computer Graphics

May 10-12, 2010
Budmerice, 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

Autodesk

Eurographics

OESTERREICHISCHE COMPUTER GESELLSCHAFT
AUSTRIAN COMPUTER SOCIETY

vrvis

Slovak Society of Computer Science

PC REVUE

The Ministry of Education of the Slovak Republic

communication solutions

CSA SYSTEMS

sféra
Grafické informačné systémy

Edited by Michael Wimmer, Jiří Hladůvka, and Martin Ilčík © 2010
ISBN: 978-3-9502533-2-0

Impressum

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

ISBN 978-3-9502533-2-0

Welcome to CESCG 2010!

This book contains the proceedings of the 14th Central European Seminar on Computer Graphics, short CESCG, which continues a history of very successful seminars. Again this year, CESCG proceedings have an ISBN (978-3-9502533-2-0) 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 is back in Budmerice castle since 2007!

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, especially Oliver Mattausch for taking care of the complete reviewing process. We are very thankful to the CESCG organizers from Bratislava, mainly Andrej Ferko, always an inspiration to CESCG; and Stanislav Stanek, Matej Novotný, Ela Šikudová, Zdenka Slobodová, Martin Samuelčík, and Pavla Ňuňuková 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 16 participating institutions and a very tight time schedule of 23 valuable student works, and two invited talks. We welcome groups from Bratislava, Slovakia; Brno (BUT and MU), Prague (CTU and KU), and Ostrava, Czech Republic; Budapest, Hungary; Bonn, Germany; Graz, and Vienna (TU and VRVis), Austria; Szczecin, Poland; Warwick, United Kingdom; Maribor, Slovenia; and Sarajevo (Univ. and SST), Bosnia and Herzegovina.

We assembled a large International Program Committee of 18 members, allowing us to have each paper reviewed by two IPC members during the informal reviewing process:

Borut Žalik	Jozef Pelikán
Jiří Bittner	Selma Rizvić
Alan Chalmers	Jiří Sochor
Andrej Ferko	Martin Šperka
Jasminka Hasić	Marc Streit
Reinhard Klein	László Szirmay-Kalos
Ivana Kolingerová	Ania Tomaszewska
Radoslaw Mantiuk	Michael Wimmer
Stephan Mantler	Pavel Zemčík

We would like to thank the members of the IPC for their contribution.

Again this year, there will be a special keynote talk held by our sponsor Autodesk: Stephen Stott will talk about "Autodesk Education Continuum - Interactive Curricula Model for Technical Excellence and Creativity". The first invited talk "Solving Vision Tasks with Variational Methods on GPUs" will be held by Horst Bischof from Institute for Computer Graphics and Vision of Graz University of Technology, Austria. The second invited talk by Roman Ďurikovič from Faculty of Mathematics, Physics, and Informatics of Comenius University, Slovakia, will be about "Simulating the Dynamics of Fluids".

The seminar is held under the auspices of the Austrian Ambassador to Slovakia, His Excellency Dr. Helmut Wessely, 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 2010:

- **Autodesk**, a world leader in 2D and 3D design software;
- **OCG**, the Austrian Computer Association;
- the **Ministry of Education** of the Slovak Republic;
- **Eurographics**, the European Association for Computer Graphics;
- **VRVis**, a research center for virtual reality and visualization in Vienna;
- **PC Revue**, a slovak computer magazine;
- **SIS**, Slovak Society for Computer Science;
- **Sféra**, graphical information systems;
- **CSA Systems**, Construction Systems Associates;
- **EEA** communication solutions.

Please note that the electronic version of these proceedings is also available at `http://www.cescg.org/CESCG-2010/`.

May 2010,

Michael Wimmer
Jiří Hladůvka
Martin Ilčík

Table of Contents

Keynote and Invited Talks

Materials

Computer Vision

Rendering

Applications

Modeling and Natural Phenomena

Data Acquisition

Color Plates

Advertisments for Sponsors of CESCG 2010

Keynote and Invited Talks

Autodesk Education Continuum – Interactive Curricula Model for Technical Excellence and Creativity

Stephen Stott

Autodesk
United Kingdom

Abstract

This presentation describes a proposal for an Autodesk Curriculum Model (ACM) that spans an Education Continuum from Secondary, Career and Technical to Higher Education. The ACM is built on an Interactive Learning Platform to engage a contemporary student audience in a learning environment focused on visual and audio communication and offers pedagogical theory combining Technical Competencies in Autodesk Software and Creative Designing Strategies.
The ACM content combines rigorous engineering theory with creative designing strategies to offer faculty and students a new pedagogical approach aligning functionality and imagination as integral elements of an exciting and engaging strategy for learning.

Solving Vision Tasks with Variational Methods on GPUs

Horst Bischof

Graz University of Technology
Austria

Abstract

This talk will present novel solutions to long standing computer vision problems by means of variational methods. We present robust methods for optical flow calculation, the correspondence problem for stereo matching, depth map integration and interactive segmentation methods. The variety of topics that can be handled by these methods demonstrate the wide applicability of variational methods.
In addition, modern graphics hardware (GPUs) allow to compute solutions to these problems very efficiently and in some cases (e.g. optical flow) even in real-time. Having real-time solutions opens several new applications areas (e.g. industrial imaging), interactive medical segmentation, etc. Some of these will be presented during the talk.

Simulating the Dynamics of Fluids

Roman Ďurikovič

Comenius University
Slovakia

Abstract

The thin film fluids can interact in the air while forming clustered structures, everybody likes to see them in the bubble show. Animation of soap bubble dynamics, formation of clusters can be handled by the dynamic surfaces colliding each other. How to animate bubbles within the fluid? We can think even about more challenging task of air bubbles within the fluid mixture. The dynamics surfaces will not help us much in this case. Fortunately, mathematical community have payed attention to the fluid dynamics and validated the model of Navier-Stokes differential equations as governing model of fluid dynamics. Unfortunately, those equations can be solved by nontrivial numerical methods. Precise numerical methods can be hardly solved in real time. Fluid dynamics governed by Navier-Stokes equations have been solved for decades but the recent trend in computer graphics is to modify the simulation to gain better controllability, in computer animation or real time fluid animation.

Materials

Real-time Fur Using GPU-based Raycasting

Martin Berger
Supervised by: Petr Kmoch

Faculty of Mathematics and Physics
Charles University
Prague / Czech Republic

Abstract

Rendering fur is an important area of computer graphics, because visually convincing fur is essential for realism of games and computer generated imagery. Our paper presents a novel technique for fur rendering based on a previously published method for grass rendering. The technique works by tracing a ray at each pixel through the fur volume with implicitly defined slices mapped with fur texture. A detailed analysis with performace tests, evaluation of applicability and comparison with another state-of-the-art technique is presented.

Keywords: Photorealistic Rendering, Programmable GPU, Fur, Volume Rendering

1 Introduction

One of the main goals of 3D graphics for many decades has been a convincing simulation and rendering of characters - humans, animals or even alien creatures. This task covers many areas of computer graphics, including model animation, facial animation, lighting models of skin and clothing and also rendering of fur or hair.

Realtime rendering of fur is an interesting area of computer graphics research because the physical properties of real fur, especially its interaction with light, are quite complex and challenging to reproduce accurately at interactive frame rates. To illustrate the incredible computational complexity of this problem, 25% of the total render time of the film *Final Fantasy: The Spirits Within* was spent on the main character's hair, according to [10].

The target applications that need to render fur in realtime are primarily games. Animals with visually pleasing fur add realism to the environment of the game. Other fields such as CGI (Computer generated imagery), where visual realism is of utmost importance, would rather prefer some of the more accurate but non-realtime methods.

Our main results presented in this paper are:

- Analysis of the grass rendering method of Habel et al. [2] along with suggestions and implementation modifications that need to be employed to adapt it to fur rendering. We report all problems encountered and present a discussion of possible solutions, along with implementation details for some of them.
- Discussion of existing real-time fur rendering methods and their comparison to the custom method. We evaluate the new method with respect to performance, realism, ease of implementation and applicable scenarios.

2 Related work

There are basically two approaches to rendering fur: geometric modelling of individual hair strands and volume based rendering.

The main drawback of rendering individual hair strands is the number of geometric primitives needed—for example, a typical bear has millions of hair strands, making this technique impractical for realtime rendering. This is, however, the technique that is often used in non-realtime applications, like CGI films.

Volume based techniques, on the other hand, can be effectively implemented on graphics card hardware and are often used in realtime applications, mainly because they offer a good compromise between rendering speed and visual quality.

One of the first volume based approaches is by Kajiya and Kay [4]. The authors introduced volumetric textures, called texels, for approximating surfaces and their properties, and rendered them using ray-tracing. Their technique produces a very high quality fur but is too slow for realtime rendering.

The authors also proposed a rather simple, *ad hoc* lighting model for hair, which later became the basis of several more sophisticated models by Goldman [1] or Scheuermann [11]. On the other hand, Marschner et al. [7] used accurate physical measurements to create a lighting model that matches the appearance of real hair and captures many effects not reproduced by the previous methods.

Arguably the most often used realtime technique for fur, textured shells, was introduced by Lengyel in [6], and subsequently improved by Lengyel et al. in [5].

In this approach, virtual hair is simulated and sampled into a volume texture in a preprocess step. The hair can be

generated by various methods that differ in their complexity and ability to control the properties of the generated hair. One of the simplest methods that comes to mind, is to randomly place hair strands, set them straight up and vary their opacity from the root to the tip. A more flexible method is given in [6], where a particle simulation is proposed that allows generating of different hair styles.

At run-time, the volume texture containing the generated fur is rendered as a series of concentric shells, each one shifted a bit from the center along the vertex normals. These shells are rendered semi-transparent using alpha blending on the GPU. An example of a model rendered with our implementation of the textured shells technique is shown in Figure 1.

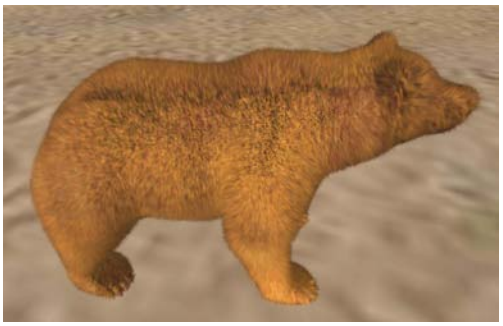


Figure 1: A model rendered with the textured shells technique (32 layers).

This method will work everywhere except at the silhouette, where the viewing angle gets very small, and the gaps between individual shells become visible. Unfortunately, the appearance near the silhouette is critical for perceiving the characteristics and structure of the fur. To address this issue, Lengyel et al. [5] proposed adding extra geometry called fins, textured with preprocessed hair texture, and rendered perpendicular to the surface at the model's edges. The opacity of the rendered fins is set to a value dependent on the viewing angle so that the fins are visible only near the silhouette.

3 Raycasting based approach to fur rendering

Grass and fur share many physical properties opening the possibility to use similar approaches to render them in real-time. Both of these natural phenomena can be characterised as a vast number of strands, which locally point approximately in the same direction. Typical difficulties with rendering any of them include aliasing errors, significant overdraw, the effects of self-shadowing and the need of primitive sorting for correct alpha blending. Due to the described similarity, it may be interesting to adapt the algorithms designed specifically for grass rendering to simulate fur and vice versa.

One such technique for rendering grass was introduced in the paper *Instant animated grass* by Habel et al. [2]. The primary goal of our paper is to render fur using a similar approach, discuss the problems that arise, and make necessary modifications to alleviate these problems.

3.1 The technique of Habel et al.

The authors propose a new approach to grass rendering. Unlike other common techniques, which often use some kind of billboarding and require new geometry to be placed into the scene, the technique of Habel et al. does not need any extra geometry and thus can be incorporated into existing material systems with little effort.

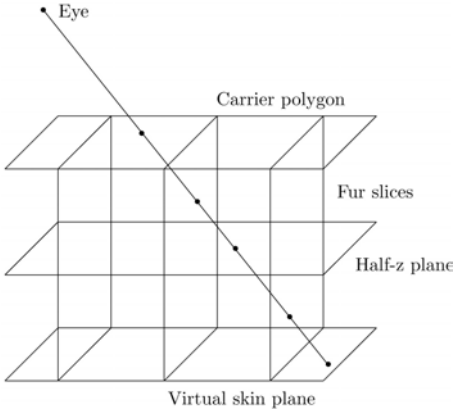


Figure 2: Visualization of ray tracing through the virtual volume containing slices with fur texture.

An outline of the technique in the context of fur rendering is as follows (see Figure 2):

- Virtual planes are defined implicitly on the model's surface. They are positioned along the u and v axes of the tangent space basis of each triangle of the model. In this way, these planes form a grid of planes perpendicular to the surface.
- In the pixel shader, a ray is traced from the viewer through the grid of planes textured with fur textures, accumulating color and opacity on its way. This method ensures, among other things, that the planes are traversed in the correct order for alpha blending.
- The ray tracing loop is terminated when the ray hits the skin plane (a virtual plane coplanar with the carrier triangle shifted by the height of virtual planes) or when a fixed maximal number of iterations is reached.
- To maintain reasonable visibility information, a depth value is calculated in the pixel shader as soon as a threshold opacity is reached.

The main assumption of this technique is that the viewer is looking at the surface mostly at grazing angles. When

viewed at perpendicular angles, the grid of planes becomes apparent. To diminish this problem, a horizontal slice, called half-z plane, is added at half of the height of the planes.

3.2 Texture map considerations

The texture with slices of fur was created manually (Figure 3), and shares the same layout with the grass slices texture described in Habel's paper. The same applies to the ground and half-z plane textures, which in our case correspond to skin and horizontal fur volume cross section, respectively. Care must be taken to make the individual fur slices unique, otherwise unwanted patterns become obvious in the rendered fur.

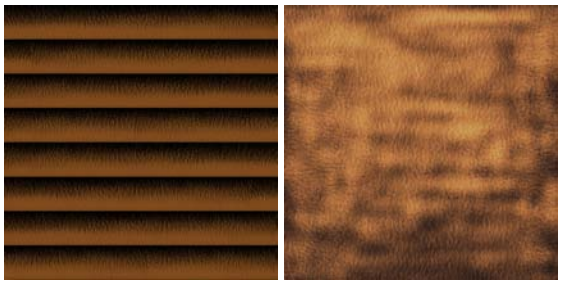


Figure 3: (a) Texture with fur slices. (b) Skin texture.

3.3 Eye direction vector

One of the first things we noticed was that the rendered fur became distorted when the camera moved close to the model's surface. When the camera was close enough for individual triangles to cover a significant area on the screen, the distortions were very noticeable and completely ruined the illusion of fur. The problem was found in the interpolation of tangent space view direction vector.

This problem can be solved by shifting the computation of the aforementioned vector from the vertex shader to the pixel shader, so that it is no longer interpolated. This modification increases the computational cost, but eliminates this problem completely.

3.4 Silhouettes

A distinctive feature of furry objects is their fuzzy appearance at the silhouette. Unfortunately, due to the nature of our method, this feature cannot be reproduced directly. The reason behind this is that the shader is executed on pixels enclosed in the projection of the model's geometry (the virtual plane space is extruded in the opposite direction of surface normals). The silhouette of this projection is composed of straight edges. Since the pixel shader can not affect pixels other than the one currently processed, the rendered fur retains straight edges at the model's silhouette. This is illustrated in Figure 4 (a).

To fix this problem, we chose to combine the basic technique with textured fins (described in Section 2). They were originally proposed to deal with a different problem (visibility of shells' structure), but as it turned out, they are able to hide the silhouette edges quite satisfactorily (see Figure 4 (b)). However, to ensure that the fins blend seamlessly with the rendered fur, the fin's texture must be similar to the texture slices used in the fur shader and also the height of the fin blades must be adjusted appropriately.

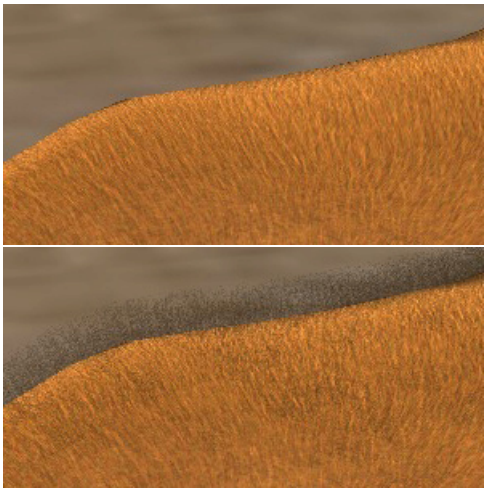


Figure 4: (a) Sharp silhouette of the original technique. (b) Original technique combined with textured fins.

The problem of silhouette appearance is not limited only to fur rendering and arises in other areas of computer graphics as well. One notable example is GPU based displacement mapping. For a detailed survey of available methods, refer to [9]. The techniques used to tackle the problem in the context of displacement mapping could provide alternative solutions to our problem. We have not investigated these possibilities further in our paper and they are left for future work.

3.5 Grid structure

The visual quality of the fur rendered with virtual planes is heavily dependent on the viewing direction. When looking at the fur at low angles, the ray tracing loop traverses many virtual planes and the visual quality is very good. However, the number of traversed planes decreases rapidly with increasing viewing angle, potentially even to zero at approximately perpendicular angles. This causes the grid structure used in the shader to become apparent, as shown in Figure 5.

The original technique was designed for rendering grass, which is rarely viewed at perpendicular angles, so this problem is not as significant there as in the case of fur rendering, where such viewing angles are very common.

We tried several approaches to address this problem. The simplest way to reduce the grid visibility is to increase the virtual plane count. Although this does not solve the

Figure 5: The grid structure is visible when looking at the surface at perpendicular angles. The model is rendered with 32 planes along each texture space axis.

problem, it makes the area with visible grid smaller. To solve this problem more thoroughly, additional modifications need to be employed.

We started with the horizontal plane mentioned in the original paper by Habel et al., positioned in the middle of the virtual plane space. It helped hide the grid on the critical places, but it also degraded the overall quality of the fur (it started having a washed-out look). Therefore we decided to modulate the color contribution of the half-z plane by a factor of $(\vec{N} \cdot \vec{V})^2$, with $\vec{N}$ being the normal vector and $\vec{V}$ the view direction vector. This factor causes the half-z plane to be visible only near the places viewed at perpendicular angles and also serves as a gradual fade-in of the visible parts.

The half-z plane does not have to be positioned exactly in half the height of the space with virtual planes. In fact, by adjusting the half-z plane position, it is possible to control its appearance to some extent. Half-z plane positioned near the top of the fur slices hides the grid but degrades the visual quality of the fur. On the contrary, rendering the half-z plane near the bottom does not help much with the grid visibility problem.

We were able to obtain quite satisfying results with dynamic half-z plane positioning. The position is given by an empiric formula we derived:

$$h = 0.9(1 - \vec{N} \cdot \vec{V}) + 0.1$$

and is calculated in the pixel shader. Note that the position is in the range $[0,1]$ with 0 being the top of the fur slices. The dot product in the formula is assumed to be properly clamped to the $[0,1]$ range. The dynamic positioning further enhances the visibility of the half-z plane in the critical places, but introduces slight visual artifacts during camera movement.

Figure 6 shows the results of the dynamic half-z positioning technique presented in this section.

3.6 Tilted virtual planes

Besides the half-z plane discussed in the previous section, we also tried some different approaches to the grid visibility problem. One of these approaches, which initially seemed very promising, is the idea of tilting the virtual

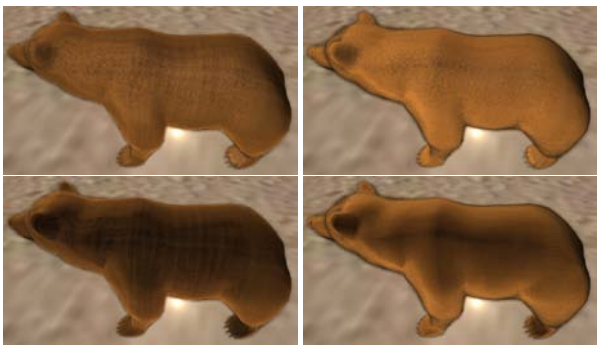


Figure 6: The effect of the half-z plane with dynamic positioning. Top left: 32 planes, half-z plane on, top right: 128 planes, half-z plane on, bottom left: 32 planes, half-z plane off, bottom right: 128 planes, half-z plane off.

planes. In the original technique, all planes are assumed to be perpendicular to the model's surface. This assumption keeps the amount of calculations inside the ray-tracing loop in the pixel shader reasonable. However, if the planes are allowed to be tilted, some parts of shader code are no longer valid, need to be generalised, and the shader complexity increases considerably.

With perpendicular planes, it is computationally inexpensive to determine the first plane intersected by the ray. Unfortunately, with tilted planes, this step is not straightforward and an exact solution would require additional calculations and non-trivial dynamic branching inside the ray tracing loop to handle the special cases that arise.

After implementing the described tilting technique, it was obvious that the visibility of the grid structure was not eliminated. Instead, this modification changed the viewing angles at which the grid was visible. Although the main problem was not solved, the tilting of the planes might still prove useful, because real fur strands rarely grow straight up.

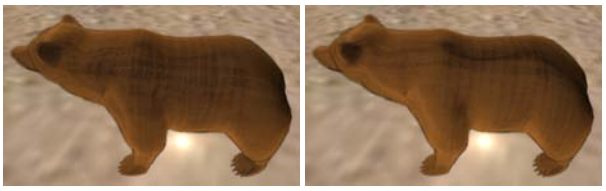


Figure 7: The effect of tilting the planes. Left: no tilting, right: planes tilted at approximately 20 degrees in both axes. Note the slant of the fur in the right image.

3.7 Problems with geometry curvature

After fixing the eye vector interpolation (see Section 3.3), the problem of severe distortions of rendered planes did not go away completely. With the camera near the surface, the rendered structure became wavy and we could

see some distortions again, though not as apparent as before.

At first, we suspected another problem with interpolation, so we tried to tessellate the mesh a lot to see whether it has any effect on the distortions. It had exactly the opposite effect from what we expected it to have. The distortions got much worse. Furthermore, we noticed that the problem was much more significant in highly curved areas.

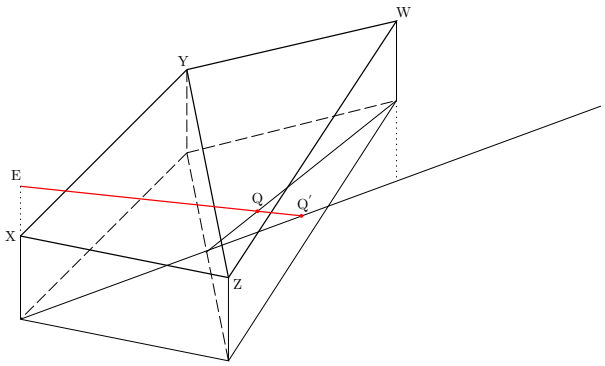


Figure 8: Schematic view of the problem with non-planar geometry. Suppose we need to find the intersection of the ray with the ground of the virtual plane volume extruded from the carrier triangle XYZ (the situation with intersecting of virtual planes is analogous). The correct intersection point is Q, since the neighbouring triangle YWZ does not lie in the same plane as the triangle XYZ. However, the ray tracing code does not account for this situation and returns the point Q' instead.

This led us to identifying the problem in the idea of extruding the volume containing the virtual planes from individual mesh triangles. During the ray tracing loop, the pixel shader has only information related to one triangle and thus cannot account for neighbouring triangles that can have different slopes. The ray traversal can eventually end up hitting planes that lie outside the volume of the original triangle (typically when the triangle is viewed at very steep angles), where the tangent space basis is no longer valid. Such a situation is depicted in Figure 8.

Unfortunately, we have to conclude that this is an inherent limitation of the original technique and there are few ways how to deal with it. This problem is not important when rendering grass, because grass usually covers some terrain with nearly coplanar individual triangles. However, fur is typically rendered on models with significant geometrical curvature.

Possible solutions to this problem include local surface approximation by quadric surfaces introduced in [8] or rendering fins on triangle boundaries as in [3].

3.8 Lighting model

To further enhance the realism of the rendered fur, we decided to implement some non-trivial lighting model. The first choice was the famous model of Kajiya and Kay [4]. The model is basically the Phong model adapted to the structure of fur (cylindrical strands). The diffuse and specular components of this model are given by:

$$\Psi_{diffuse} = k_d r \sin(T,L)$$
$$\Psi_{specular} = k_s((T \cdot L)(-T \cdot V) + \sin(T,L)\sin(T,V))^p,$$

where $k_{d,s}$ are the diffuse and specular reflection coefficients, r is the radius of the hair strands, T is the tangent vector pointing from root to tip, L is the light vector, V is the eye vector and p is the standard Phong specular exponent. For derivation of these formulas, refer to [4].

This model captures only a small part of the light scattering process of real hair strands. The biggest limitation is the fact that the model deals only with first-order light reflection. However, real hair exhibits both reflective and transmissive behavior. An accurate solution would have to consider scattering of light onto other hair strands and also onto the underlying skin. These light interactions result in physically complex visual phenomena such as secondary highlights observed on real hair (see [7] for a detailed discussion of these phenomena).

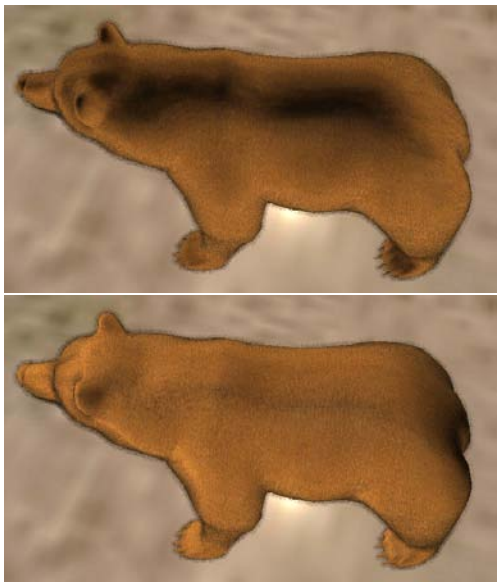


Figure 9: The lighting model (a) without and (b) with shifting of the tangent vector T. The light is positioned a bit to the left and above the bear model.

To extend the Kajiya-Kay lighting model, we added the calculation of a directional attenuation factor, as described in [1]. Adding this factor increases the directionality of the hair. Moreover, the relative reflectivity (backward scattering) and transmissivity (forward scattering) is parametrized by two factors, $\rho_{reflect}$ and $\rho_{transmit}$, that can be used to tune the produced result.

After implementing the described lighting model, the visual results were not as good as we expected. The main reason behind this is that the hair strands are positioned

straight up on the model. This is in contrast with real fur, which usually has some slant. As a quick and dirty solution, we add a tangential component to the normal vector on the surface (this vector corresponds to T in the equations). The length of the component is modulated by a sample from a noise texture to make the lighting a bit softer. The effect is shown in Figure 9.

4 Results

4.1 Performance analysis

The custom method we developed can be thought of as an alternative to the textured shells technique. Both techniques are capable of producing high quality fur, but with rather different computational requirements.

With textured shells, rendering a model with 32 layers causes the model to be submitted to the renderer 32 times. This may pose a burden both on the geometric (in case of a high polygon count model) and pixel shading stages of the rendering pipeline. However, the simplicity of the vertex and pixel shader programs makes rendering of a large number of layers feasible.

On the other hand, with the technique presented in our paper, the geometric complexity is independent of the number of virtual planes or the maximum passes of the ray-tracing loop. Also, due to the nature of the technique, the overdraw is basically zero. The complexity of this technique lies in the long pixel shader full of dynamic branching, where a ray-tracer is implemented.

The performance of this method is expected to be influenced the most by the average number of iterations of the main ray-tracing loop. There are two parameters that can affect this number: the virtual plane count and the ray-tracing loop limit. With small virtual plane counts, most of the rays intersect only a few planes before hitting the ground and breaking the loop. The maximum number of iterations is not the limiting factor in this case, because it affects only the rays that penetrate the virtual plane space at the silhouette. However, this factor becomes critical when the planes are laid out densely. The dependency of these factors is discussed in detail in the next section.

4.2 Tests

Let us examine the performance of our method in detail. For reference, we have included some tests with the textured shells technique applied to the same geometric model. In each test, the model was rotated in front of the camera for a fixed amount of time and average FPS was recorded. The individual tests were:

Tests	Shader	Resolution	planes/layers	loops
T1 - T3	Our	1600×1200	32, 64, 128	5
T4 - T6	Our	1600×1200	32, 64, 128	10
T7 - T9	Our	1600×1200	32, 64, 128	20
T10	Our	1920×1200	128	20
T11	Shells	1920×1200	32	-
T12	Shells	1600×1200	32	-
T13	Shells	1600×1200	16	-

The results of the tests, obtained on a desktop PC with Intel Core2Duo 3.0 GHz CPU, 4GB RAM and ATI Radeon HD4850 graphics adapter with 512MB of VRAM, are given in the following table:

Test	Average FPS	Test	Average FPS
T1	297.6	T8	151.6
T2	297.3	T9	139.1
T3	305.9	T10	121.7
T4	224	T11	99.9
T5	207.7	T12	108.4
T6	206.9	T13	190.6
T7	164.2		

These results clearly confirm the hypothesis that the maximum number of iterations in the ray-tracing loop is the most important parameter from the performance point of view.

Tests T9 and T12 can be chosen for visual comparison of both techniques with high quality settings. Details from the rendered images are shown in Figure 10. As can be seen from the image, the textured shells produce a more fluffy fur. The novel technique produces a fur that is optically lower but the fur structure is more detailed from a close-up look. With these settings, our method runs slightly faster (a few tens of FPS).

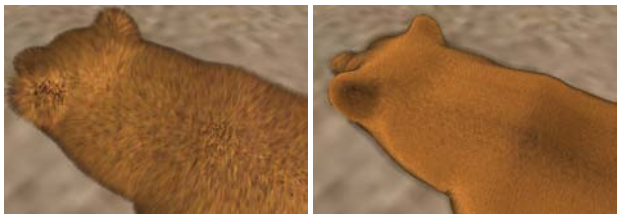


Figure 10: (a) The technique with settings from the test T12. (b) The technique with settings from the test T9.

4.3 Applicable scenarios

The question, which of the two methods to implement, is more of an artistic one. There are, however, some facts that might favor one method over the other. The fur rendering method presented in our paper is generally suitable for shorter fur types because of the behavior at the silhouette. Long fur would probably be more realistic with the textured shells technique.

The novel method integrates well into existing rendering pipelines. There are, however, some changes beyond the shader compilation and usage that need to be taken care of when implementing this technique. The most important is the generation of fins' geometry. This step is required with textured shells as well, though. The amount of work needed for authoring the textures used in the shaders is comparable for both techniques.

The target shader model of the HLSL compilation must be at least the version 3.0, as the previous versions do not support dynamic branching. This places a restriction on the hardware capable of running these shaders. This is in contrast to textured shells, whose shaders utilize only a very basic feature set enabling the use of this technique even on older hardware not capable of shader model 3.0.

Another factor, that should be taken into consideration, is the memory usage. Textured shells, being a volumetric method, consume a significant amount of memory. The proposed solution has much lower requirements, as it uses only four textures of reasonable size.

5 Conclusion

5.1 Summary

We have introduced the topic of fur rendering and discussed possible approaches along with their respective advantages and limitations. We have put emphasis on describing the textured shells technique, because it was used for comparison with the novel method.

As the main goal of the paper, we have presented a new fur rendering technique, utilizing an approach introduced in the context of grass rendering in [2] and then proceeded to a discussion of problems we encountered. We have presented solutions to most of these problems and, where possible, stated the impact of the modifications on the source code complexity and performance. We have also presented some ideas for implementation of a non-trivial lighting model. Finally, we discussed the pros and cons of the novel technique, and presented some performance tests with a comparison with the textured shells method.

5.2 Future directions

The method, as it has been described, does not include any form of LOD (Level of detail). Developing some kind of LOD could prove interesting as it would widen the range of applications of this technique. We have not investigated the possibilities in detail but we think that the parameter controlling the maximum number of ray-tracing loop iterations could be used to create some LOD scheme.

Furthermore, the last problem described in Section 3.7 needs a solution. At the end of that section, we give references to papers containing work related to similar problems. We believe that a viable solution to our problem could be derived from them.

Finally, the lighting model could be improved to account for self-shadowing and to better handle the uniform fur direction that results from perpendicular virtual planes.

6 Acknowledgements

I would like to thank Petr Kmoch for his guidance during the creation of this work.

References

[1] Dan B. Goldman. Fake fur rendering. In *Proceedings of SIGGRAPH '97*, pages 127–134, New York, NY, USA, 1997.

[2] Ralf Habel, Michael Wimmer, and Stefan Jeschke. Instant animated grass. *Journal of WSCG*, 15(1-3):123–128, January 2007.

[3] S. Jeschke, S. Mantler, and M. Wimmer. Interactive smooth and curved shell mapping. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 351–360, 6 2007.

[4] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *Proceedings of SIGGRAPH '89*, pages 271–280, New York, NY, USA, 1989.

[5] J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe. Real-time fur over arbitrary surfaces. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 227–232, New York, NY, USA, 2001.

[6] J. E. Lengyel. Real-time hair. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 243–256, London, UK, 2000.

[7] S. R. Marschner, H. W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan. Light scattering from human hair fibers. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 780–791, New York, NY, USA, 2003. ACM.

[8] M. M. Oliveira and F. Policarpo. An efficient representation for surface details. Technical Report RP–351, UFRGS, January 2005.

[9] L. Szirmay-Kalos and T. Umenhoffer. Displacement mapping on the GPU - State of the Art. *Computer Graphics Forum*, 27(1), 2008.

[10] Scheuermann T. *Hair Rendering and Shading*. GDC presentation, 2004.

[11] Scheuermann T. Hair rendering and shading. In Wolfgang Engel, editor, *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 239–250. Charles River Media, 2005.

Time-Varying BTFs

Tobias Langenbucher, Sebastian Merzbach, David Möller,
Sebastian Ochmann, Richard Vock, Welf Warnecke, Michael Zschippig
Supervised by: Martin Rump

Institute of Computer Science
Rheinische Friedrich-Wilhelms-Universität Bonn
Bonn/Germany

Abstract

There have been several approaches to model and capture time-varying materials. Modeling approaches provide good results but are sometimes hard to apply because underlying processes are not yet understood or very complex. In this paper, we present a data-driven approach to record aging effects of metal and car paint with the help of Bidirectional Texture Functions. BTFs precisely capture spatially varying reflectance properties of a given material. However, once captured, one cannot change the appearance of a material that ages. Instead, we measure at several distinct aging steps and combine this information into a time-varying BTF which allows the user to interpolate between different stages of the aging process.

1 Introduction

One goal of computer graphics is to create photo-realistic images. In order to accomplish this goal, it is necessary to take reflectance properties of materials into account. These properties depend on color, shininess, translucency and inner structure of a given material.

Additionally, almost all materials age, e.g. through corrosion, scratches, or bleaching, leading to a change of their visual properties over time. It is important to capture these effects as well since they strongly contribute to the realism of rendered materials and being able to smoothly interpolate the age of materials offers vast possibilities for industrial, artistic and scientific applications. This can be used for example to examine what a product will look like after months or years of use.

This paper is a practical report about our work measuring and rendering TV-BTFs (Time-Varying Bidirectional Texture Function). We chose time-varying BTFs because they capture these effects very well and have not been analyzed before.

Solving the rendering equation by Kajiya [4] for any given scene is the primary challenge in realistic rendering. One important component of this equation is a function that returns the amount of reflected light given the direction of incoming and outgoing light. A popular approach to represent this function is called BRDF (Bidirectional Reflectance Distribution Function). The BRDF is a function that describes how much light from an incoming angle is reflected to an outgoing angle. It does not take subsurface-scattering into account and is restricted to homogeneous surfaces only. Some other groups have already worked on Time-Varying BRDFs [8]; those however, unlike BTFs, are not capable of capturing visual properties of whole patches of a material.

There are two popular ways to generate a BTF. The first one uses phenomenological and analytical models, the second one is a data-driven approach. One big disadvantage of the model-based concept is the fact that you have to create a very detailed and physically accurate model of the material, which is often excessively time-consuming or even impossible. We therefore use a data-driven approach, measuring the BTF of a material using a dome consisting of 151 digital consumer cameras. An in-depth description of this setup is given in chapter 4.1.

We examined our approach by letting a simple metal plate rust and scratching a sample of green car paint. To get time-varying results, we measured several times with our increasingly aged materials and combined the data into a single Time-Varying BTF. Using this TV-BTF, it is possible to interpolate over the lifetime of the material.

In the following chapters we will describe the basics of rendering techniques, global illumination, capturing, compression and decompression of Time-Varying BTFs. In chapter 6 we will present some images of basic geometric figures rendered with our Time-Varying BTF.

2 Basics

To achieve photo-realistic rendering, one has to measure reflectance behavior of materials. This behavior is characterized by how a ray of light with wavelength λ_i, hitting the surface in a point $\mathbf{x_i}$ in direction $\omega_\mathbf{i}$ at time t_i, travels through the material and leaves at another point $\mathbf{x_r}$ in direction $\omega_\mathbf{r}$ at time t_r and with possibly altered wavelength λ_r. As shown in Figure 1, this leads to a twelve-dimensional function (the points $\mathbf{x_i}$, $\mathbf{x_r}$ have two spatial coordinates and the directions $\omega_\mathbf{i}$, $\omega_\mathbf{r}$ are represented by two angles).

It is, however, computationally way too complex and thus currently impractical to fully describe arbitrary ma-

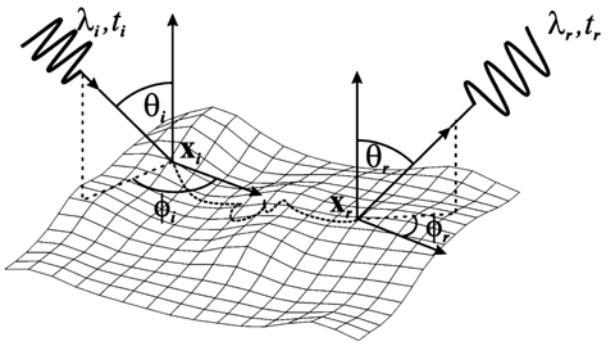


Figure 1: The twelve-dimensional function describing light transport in materials. [5]

terials because of the high dimensionality of a physically correct surface reflectance function. Measuring or storing this information is beyond today's hardware capabilities. As a result, several simplifications are used to reduce the parameter count. Phenomena such as fluorescence (change of wavelength during reflection) or phosphorescence (re-emission of absorbed light after a certain amount of time) are usually neglected, i.e. $\lambda_i = \lambda_r$ and $t_i = t_r$. Furthermore, the electro-magnetic spectrum is usually discretized into three wavelengths, i.e. red, green and blue light.

These simplifications lead to the time-independent eight-dimensional BSSRDF (Bidirectional Surface Scattering Reflectance Distribution Function). The BSSRDF is still quite complex and hence, further reductions have to be applied. A simplification of the BSSRDF is the BTF (Bidirectional Texture Function) which does not fully account for subsurface scattering.

$$\begin{aligned} &\mathrm{BTF}(\mathbf{x_r}, \theta_i, \phi_i, \theta_r, \phi_r) \\ &= \int_{\mathrm{Surface}} \mathrm{BSSRDF}(\mathbf{x_i}, \mathbf{x_r}, \theta_i, \phi_i, \theta_r, \phi_r) d\mathbf{x_i} \end{aligned} \quad (1)$$

One can easily see in the above equation that the BTF only depends on *one* position. It accumulates light that is scattered in the material from neighboring regions. Most measurement setups that are used to capture BTFs of planar materials (c.f. chapter 4.1) usually gather the subsurface scattering part; it is, however, inseparably contained in the measurements.

Even further simplifications include the four-dimensional BRDF (Bidirectional Reflectance Distribution Function) that discards any positional information and only depends on incident and outgoing directions.

$$\mathrm{BRDF}(\theta_i, \phi_i, \theta_r, \phi_r) = \frac{\int_{\mathrm{Surface}} \mathrm{BTF}(\mathbf{x_r}, \theta_i, \phi_i, \theta_r, \phi_r) d\mathbf{x_r}}{||\mathrm{Surface}||_{BRDF}} \quad (2)$$

Rendered with a BRDF, a material shows the same reflectance behavior over the whole surface.

3 Previous Work

In the field of time-varying materials, BRDFs, variable textures and aging-models have already been analyzed. [2] provides a good overview of this.

Time-varying BRDFs: At the CAVE laboratory at Columbia University, Sun et al. have created a database for time-varying BRDFs. They measured BRDFs of aging materials every 36 seconds. With this setup, they obtained a wide range of data about different time-varying materials. In the next step, they fit BRDF functions to this data and extracted the parameters. After that, they used these parameters to develop time-varying BRDFs. In [8], Sun et al. focused on drying materials and accumulation of dust.

Time-varying textures: In [3], Enrique et al. captured several hundred pictures of a given material over time. They extended the texture function with a time parameter. Afterwards, they were able to texture a material at any point of time.

Aging-models: To model aging effects on metallic patinas, Dorsey et al. use a collection of operators applied to a layer model. Each of these operators represent a different weathering effect. The result is a surface of different thickness and therefore varying reflectance properties [1].

4 BTF-Pipeline

4.1 Acquisition

A bidirectional texture function (BTF) is a six-dimensional function which provides information about the appearance of a material depending on the position on the object's surface as well as the viewing and lighting directions. Practical measurement of this function on a material sample, or rather an approximation thereof, is performed by taking several thousand pictures from different, discretized viewing and lighting angles.

We obtain the required images by using a hemispherical camera dome by Sarlette et al., consisting of 151 consumer cameras as described in further detail in [7]. All cameras simultaneously shoot photos of a material sample placed in a fixed position inside the dome while the integrated flash light of one of the cameras illuminates the sample. This process is repeated for each lighting direction which yields photos for all 22,801 combinations of camera and lighting positions.

The setup allows for rapid, automated measurement of the materials thanks to the highly parallelized process as well as a relatively simple post-processing phase due to the rigid camera setup with known extrinsic (position and orientation of the camera) and intrinsic (focal length, etc.) camera parameters.

4.2 Geometric Calibration

To determine these parameters, we use the same calibration technique as presented in [7], which uses a planar cal-

ibration object with 121 well-known LED features. The use of active markers compared to passive ones leads to a more accurate and robust detection.

We first measure without optical zoom. As we can make the assumption that the intrinsic parameters are the same, the extrinsic parameters can be estimated using Zhang's camera calibration algorithm [9]. Afterwards, the intrinsic parameters are calculated zoom step by zoom step until the measurement zoom step is reached.

4.3 Radiometric Calibration

To compensate slight differences of the camera sensors and the spectral distribution of the flash lights, four reflectance targets with different albedo values and known reflectance behavior are positioned near the target.

Using these markers, white-balance factors are calculated to achieve consistent color reproduction and to calculate the effects originating from spectral differences in flash light. In a post-processing step, these effects are eliminated by simply multiplying measured values with white-balance factors.

4.4 Image post-processing

After the raw images have been acquired they need to be transformed into a common coordinate system where all images have the same orientation and size.

We can compute the necessary transformation to reproject the region of interest within each image to the desired coordinate system by using the known extrinsic camera parameters as well as corner markers in the images. Small inaccuracies during measurement are compensated by searching a small region around the assumed corner positions and using the adjusted positions for the transformation.

For decent rendering results, we need high dynamic range images. However, one measurement pass from the consumer cameras with some given ISO speed (i.e. CCD-sensitivity) and flash light intensity only provides low dynamic range images. Thus, we perform multiple passes with different settings for the ISO speed and flash intensity which we later merge into one HDR image for each combination of the camera and light directions. The relative position of the single LDR steps in energy space is extracted from the white markers.

4.5 Compression and Decompression

Storing the resulting images uncompressed - assuming an image size of 768×768 pixels each - takes up approximately 150 GB of storage capacity, which is not feasible for later usage in a shader.

In order to use the captured BTF data for fast rendering, a decent compression algorithm is needed which has three key requirements:

1. It should allow for fast random access to a single pixel.
2. It should exploit recurring patterns and similarities in the data in order to achieve high compression rates, feasible for GPU calculations.
3. It should compress the data in reasonable time.

While the latter (as an offline computation) is a less critical requirement than the other two, the time spent compressing should meet at least decent practical considerations.

Traditionally, algorithms that fit these requirements project the data into a lower-dimensional space by building a set of orthogonal base vectors and dropping the least significant ones.

Since the uncompressed TV-BTF data is a seven-dimensional tensor, common ways to compress it involve either tensor or matrix decomposition, i.e. unfolding the tensor into a matrix beforehand. Tensor decomposition techniques allow for exploitation of patterns and similarities in each dimension separately and hence tend to achieve higher compression ratios than a matrix decomposition which is restrained to one or two dimensions.

However, known tensor decomposition algorithms take too much time to precompute and are thus not applicable. We therefore chose a matrix decomposition approach by using principal component analysis (PCA) to project the BTF data into a theoretically optimal orthogonal base. A PCA currently fits our requirements stated above best:

1. Access to a single value of the BTF data involves only a scalar product of 2 c-dimensional vectors - c being the dimension of our orthogonal subspace (i.e. the number of principal components kept).
2. Considering an intelligent unfolding into a matrix, a PCA builds an "optimal" subspace by using orthogonalized directions of the highest covariance, thus exploiting similarities and patterns in our data.
3. By using an EM-PCA algorithm introduced in [6] we are able to compute only the c most important components in a fast way without losing computation time on components we would drop later anyways.

In order to calculate the PCA, a matrix representation of the measured data is needed. For the PCA to work properly, i.e. generate quickly descending eigenvalues and hence allow for good compression, similarities in the data should be placed close to each other in the matrix. The expected variance between adjacent pixels is less than the expected variance between one pixel under different lighting conditions.

To meet these requirements, we decided to store each color channel for each time step in separate matrices, compressing each of them using PCA. Therefore, one time step consists of three matrices. The scheme works as follows:

Each column of each matrix holds the values of one color channel (red, green or blue) of one entire HDR-image. All images are written into the matrix sorted by light and view directions (see Figure 2). The dimension of the resulting matrix is $(w \cdot h) \times (v \cdot l)$, where w and h are the width and height of the HDR-images, v is the number of view directions and l is the number of light directions.

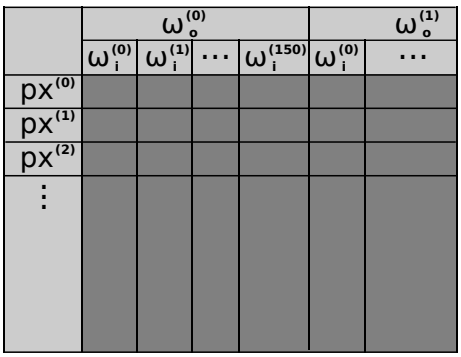


Figure 2: BTF data matrix layout for one color channel.

Regarding the large amount of data involved, composing the BTF data matrix in this way involves the problem of an out-of-core matrix transposition, since only a limited amount of images can be kept in memory simultaneously. However, this computation is a very time-consuming task if the matrix is big. As IO (reading images from hard disk, writing the data to the matrix file) is the bottleneck of this task, we used an algorithm, that writes the matrix in chunks that fit into memory (see Figure 3). This minimizes the number of times each image has to be read.

Creating a single matrix including all measured time steps and thus using a global compression for one large matrix would significantly increase compression time and memory usage (beyond what we are able to compute using our GPU-implementation in decent quality). For this reason, we chose to use separate matrices for each time step and interpolated individual pixel values from these matrices linearly in the shader. In the case of rusting metal, color and geometric properties of the material change dramatically over time so that a global compression would not provide any reasonable benefits concerning compression ratio and quality. Using separate matrices, we achieved a sufficient compression ratio to use for realtime-shading (in case of the metal we cut down matrix size from 17.5 GB to 78.7 MB).

5 Measurement

The following table gives a quick overview of the measurement setup. In detail explanation follows in the next two sections.

```
imgList = getSortedHDRImageFilenames();
largeFileR = reserveMemory();
largeFileG = reserveMemory();
largeFileB = reserveMemory();
written = 0;
// chosen according to resulting memory usage:
numScanlines = 20;

blocks = emptyListOfImages();
while (written < imgHeight) {
  // load images into RAM
  for (i = 0; i < sortedList.size(); ++i) {
    // get i'th block of scanlines
    blocks[i] = readImageBlock(
      imgList[i],
      // first scanline
      written,
      // last scanline
      min(imgHeight, written + numScanlines)
    );
  }

  // write data into matrix file pixel-wise
  for (px = 0; px < imgHeight*imgWidth; ++i) {
    for (img = 0; img < blocks.size(); ++i) {
      // write R, G and B values
      largeFileR.write(blocks[img][px][R]);
      largeFileG.write(blocks[img][px][G]);
      largeFileB.write(blocks[img][px][B]);
    }
  }

  written += numScanlines;
}
```

Figure 3: Pseudo code for writing the BTF data matrix.

light directions	151
view directions per flash	151
iso levels	4
aging levels	4
total pictures taken	364,816

5.1 Metal

To measure visual properties of metal rusting over time, we took measurements of a metal plate in a progressively rusted state. Between the measurements in the dome we sped up the aging process using a rusting-chamber as schematized in Figure 4. The bottom was filled with warm (approx. 28°C), salted water which was tempered and circulated with an aquarium heater and pumps. The probe stood nearly upright (with an angle of about 75°) on a support frame. Because the rusting process is greatly accelerated by the presence of acid we placed a small basin filled with vinegar in front of it. To counteract water drops on the probe, we heated the roof above with a lamp, so that the vapor could not condense in this area.

Whenever we considered a new stage of aging to have been reached, a new measurement was taken. The (entirely experimental) time intervals used for the aging of the material are listed in Table 1. We measured four

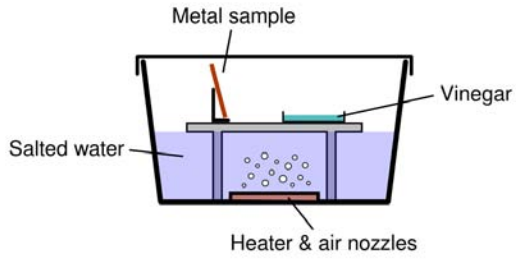


Figure 4: Our setup to accelerate the rusting process between measurements.

states of the rusted material, each measurement consisting of four different exposure time and flash settings for later HDR combination. Thus we acquired a total of $151^2 \cdot 4^2 = 364,816$ individual images, resulting in about 1,075 GB of storage space for the unprocessed shots.

Figure 5 shows photo-montages of the different rusting states assembled from raw measurement footage of the metal experiment. The pictures from the four different rusting states have been combined and faded from left to right, showing the different rusting states.

Measurement #	Time rusted
1	0 min
2	+10 min
3	+30 min
4	+30 min

Table 1: Time intervals for the rusting process.

5.2 Car Paint

Before taking initial measurements of the car paint, we had to clear it of pre-existent scratches by applying common car polish. We then recorded the pictures for a BTF. Afterwards, we added new scratches with a coarse and raspy sponge that is normally used to wipe insect remnants off your windshield. We tried to move the sponge in circles to scratch the surface evenly along all directions. Utmost care had to be taken not to add too many new scratches because the procedure is irreversible. After each step of scratching, we carefully determined the amount of scratches by taking photos with a flash light, because the scratches are hardly visible in a diffuse lighting as it is common indoors.

Just like the patina material, the car paint shows highly specular reflections, so we had to use four different ISO speeds and flash intensities to get qualitatively acceptable HDR pictures. We used five different stages of altered material, leading to a total of 364,816 pictures of the material and about 1 TB storage space.

A problem exists in our measurement setup with highly reflective materials, as there are reflections of cameras from the opposite side of the dome from low angle shots. This problem and its solution are discussed in [7]. However, we did not apply this solution because these artifacts are negligible with our materials.

6 Results

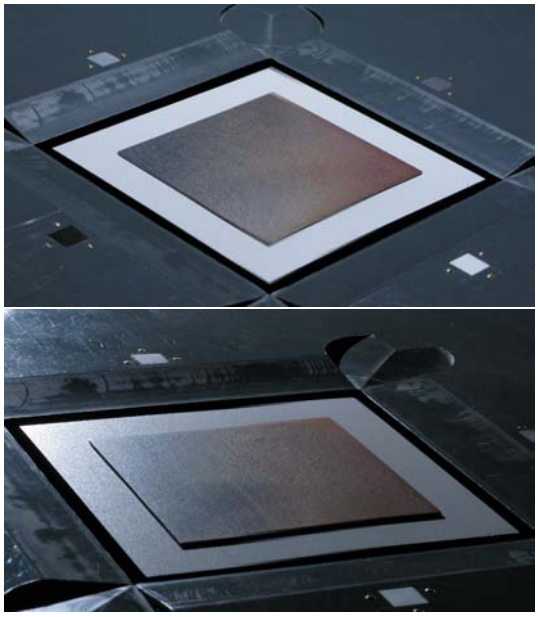


Figure 5: Montages of raw measurement footage.

We were able to compress more than one terabyte of raw image data to a few megabytes. Using only the 100 most significant components in the PCA, we achieved compression ratios of $42,000 : 1$ for the car paint.

Figure 6 depicts path tracer renderings of a sphere with the different rusting states and linear interpolations between them applied to the surface.

In Figure 7, three actual photographs of the first, second and fifth measurement are shown. The third and fourth picture were created by linear interpolation between the real data. In Figure 8 you can see actual renderings of the first and last aging stage.

Interpolation works well on the data and it can be used for TV-BTFs. One can argue that linear interpolation is unsuitable for our approach. Scratches appear suddenly and do not fade in smoothly, as it occurs when interpolating linearly. However, these artifacts are hardly visible and handling this problem properly is beyond the scope of this paper.

7 Conclusion

The results show that our approach can be used to render images of time-dependent materials. The data can be com-

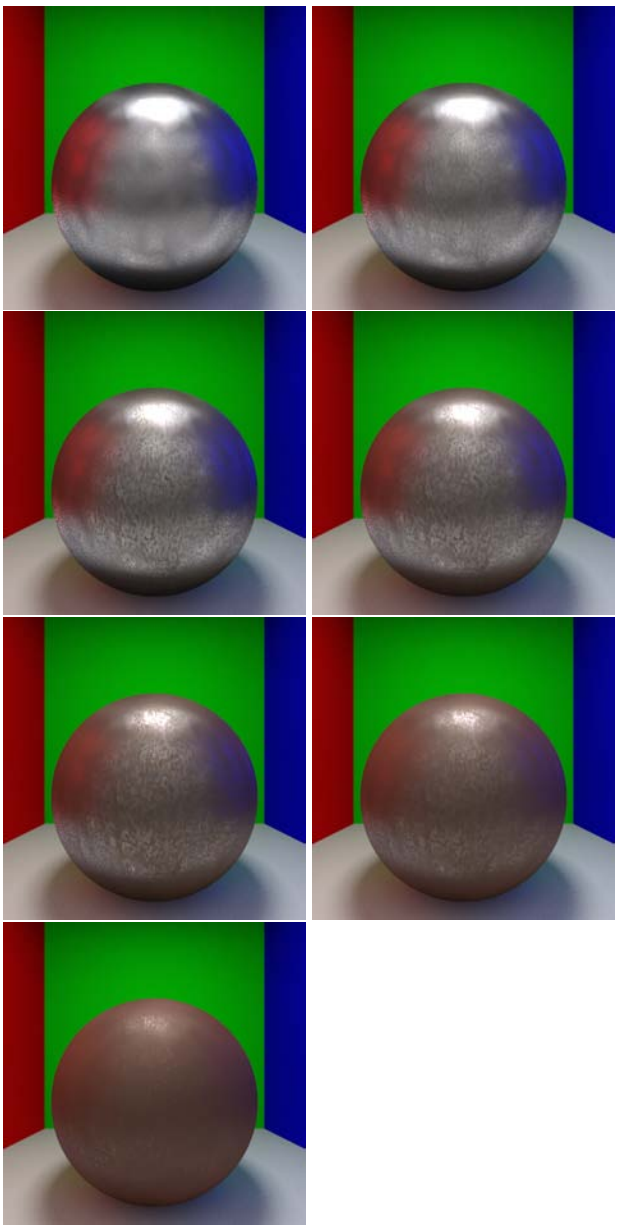


Figure 6: Rusted metal renderings. Left column: Rendered images of the four measured rusting states; right column: linear interpolations between the neighboring states within the shader.

pressed very well and the interpolated aging levels appear realistic.

There are however some problems that have to be dealt with in future projects. Small errors in measurement can lead to big errors in the resulting BTFs, e.g. scratches on the car paint do not match in different aging steps. This problem is caused by the sample holder not being fixed in the dome. It is possible for the sample to lie in a slightly different angle or position relative to the flashlights in subsequent shots, resulting in different highlights on the scratches. This effect can not be compensated by

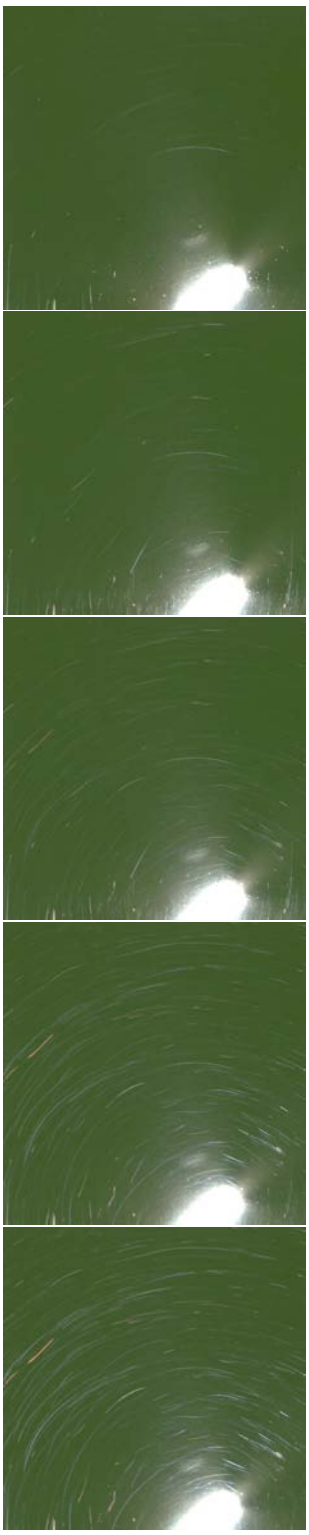


Figure 7: Montages of raw measurement footage, from top to bottom: First measurement, second measurement, followed by two interpolations and the last measurement

the rectifications used so far. It is, however, hardly visible because of the small scale of these highlights.

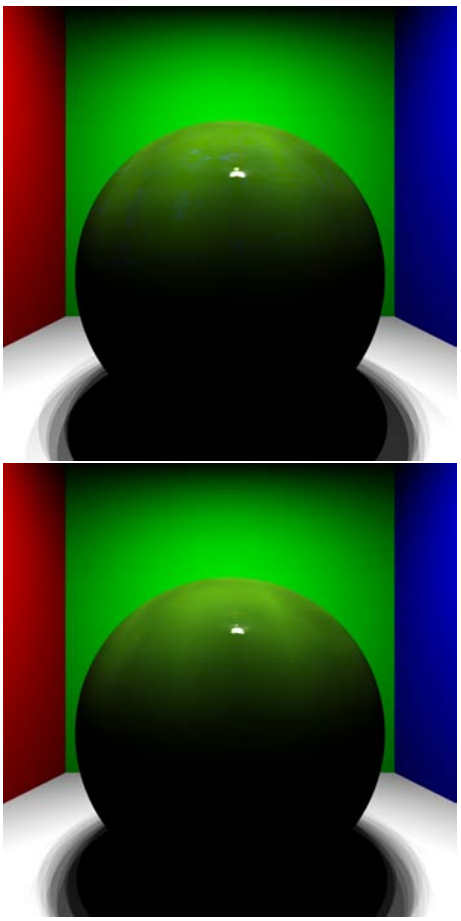


Figure 8: Actual renderings of the first and last aging stage of the car paint material.

8 Acknowledgment

We would like to thank our project supervisor Martin Rump, as well as Ralf Sarlette and Professor Dr. Reinhard Klein who allowed us to gain better knowledge of BTFs and insight into the research activities and work in our university's computer graphics department.

References

[1] Julie Dorsey and Pat Hanrahan. Modeling and rendering of metallic patinas. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 387–396, New York, NY, USA, 1996. ACM.

[2] Julie Dorsey, Holly Rushmeier, and François Sillion. Advanced material appearance modeling. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–145, New York, NY, USA, 2008. ACM.

[3] Sebastian Enrique, Melissa Koudelka, Peter Belhumeur, Julie Dorsey, Shree Nayar, and Ravi Ramamoorthi. Time-varying textures: definition, acquisition, and synthesis. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 130, New York, NY, USA, 2005. ACM.

[4] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, 1986.

[5] Gero Müller, Jan Meseth, Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Acquisition, synthesis and rendering of bidirectional texture functions. *Computer Graphics Forum*, 24(1):83–109, March 2005.

[6] Roland Ruiters, Martin Rump, and Reinhard Klein. Parallelized matrix factorization for fast btf compression. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 25–32, March 2009.

[7] Martin Rump, Gero Müller, Ralf Sarlette, Dirk Koch, and Reinhard Klein. Photo-realistic rendering of metallic car paint from image-based measurements. *Computer Graphics Forum*, 27(2), April 2008.

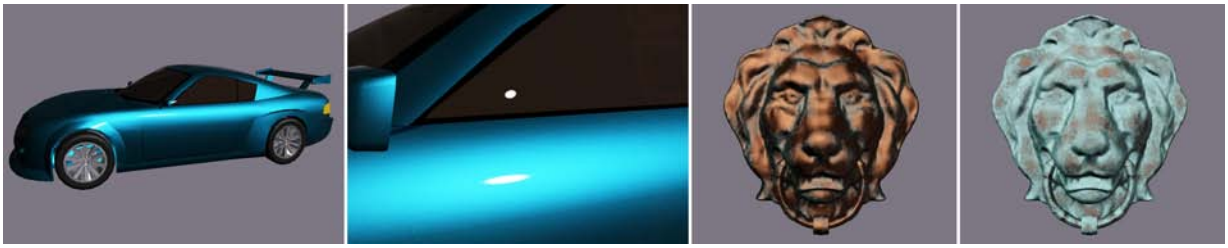
[8] Bo Sun, Kalyan Sunkavalli, Ravi Ramamoorthi, Peter Belhumeur, and Shree Nayar. Time-Varying BRDFs. In *Eurographics 2006 Workshop on Natural Phenomena*, Sep 2006.

[9] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, 2000.

Layered Materials in Real-Time Rendering

Oskar Elek

Faculty of Mathematics and Physics
Charles University
Prague / Czech Republic



Abstract

Today's games and other real-time 3D applications often use only basic empirical models for modelling the appearance of materials and rely on complex geometry and texturing to make them more visually appealing. In this paper we explore the possibilities of bringing more physically plausible models to real-time 3D graphics.

We do this by implementing the layered BRDF of Weidlich and Wilkie on GPU. This model utilizes the well-known Torrance-Sparrow and Oren-Nayar microfacet models. We show how to make this layered model useful for real-time rendering through various optimizations. Then we derive two specialized models based on this basic layered model. These two models attempt to simulate the appearance of metallic car paints and metallic patinas.

Keywords: surface reflectance models, appearance modelling, layered materials

1 Introduction

Our environment contains a large variety of objects and materials with surfaces composed of multiple layers, for instance coated ceramics and plastics, varnished and patinated metals, organic tissues etc. Rendering such materials in real-time applications is difficult — traditionally used empirical BRDFs, such as the Phong reflectance model, are not capable of reproducing the appearance of most of them, and possibility of utilization of measured BRDF or BTF data is limited on GPU. There is a need for a simple analytical BRDF that can reproduce the appearance of layered materials in interactive applications.

This paper extends the work of Weidlich and Wilkie [21], who presented a simple analytical physically-based BRDF suitable for rendering layered materials. Their model is based on combination of commonly used BRDFs, such as Torrance-Sparrow [20] and Oren-Nayar [15] reflectance models. We show how to transfer their work into the environment of real-time evaluation in fragment shaders, along with various convenient optimizations. We then demonstrate the potential of this model by employing it in two specialized models for rendering metallic car paint and copper patina.

The paper is organized as follows: we first give an overview of related work in the field and review the layered model of Weidlich and Wilkie. Then we discuss a real-time adaptation of this model, along with two specialized models for rendering metallic car paint and metallic patinas. Finally, we measure the performance of our implementation and discuss its possible utilization in real-time applications.

2 Background and Related Work

2.1 Microfacet Reflectance Models

Analytical reflectance models used in 3D computer-generated imagery can be generally divided between two groups: *empirical* and *physically-based*. Empirical models are based on direct observation and therefore are usually physically implausible (except for the limit cases — ideal diffuse and mirror reflectors). The most common BRDFs from this group are Phong [16] model and its modification by Blinn [1], both widely used in real-time rendering applications. The prevalence of these models in real-time rendering is due to their low computational requirements and fairly good reproduction of overall object appearance.

In contrast to these, physically-based BRDFs model reflectance properties of materials from first optical principles, which leads to a more plausible appearance of materials. A BRDF is physically plausible, if it conserves energy (i.e. albedo is always at most 1), obeys Helmholtz reciprocity principle and is non-negative. To retain a closed analytical form, they often use a statistical surface representation, instead of an explicit one. Surfaces are represented by statistically distributed tiny microfacets or V-shaped cavities. We will discuss two physically-based BRDFs, namely the Torrance-Sparrow [20] and Oren-Nayar [15] reflectance models.

The Torrance-Sparrow reflectance model builds on the as-

sumption that the material surface consists of microscopic V-shaped cavities that behave like perfect mirrors with Fresnel reflectance. The amount of reflected light f_r is defined as

$$f_r = \frac{FDG}{\pi(N \cdot L)(N \cdot V)} \quad (1)$$

- $F(\beta, n, \kappa)$ is the Fresnel term. It expresses the reflectance coefficient of each individual microfacet. β is the angle between incident light direction L (or view direction V) and half vector H, n and κ are real and imaginary components of the material's index of refraction (IOR). Note that F is wavelength-dependent. The full Fresnel term equations can be found in [2] or [7].
- $D(\alpha, m)$ is the microfacet slope distribution function representing the amount of microfacets oriented towards the observer. α is the angle between half vector H and surface normal N, $m \in \langle 0,1 \rangle$ is the surface roughness parameter. Small values of m represent very smooth surfaces, while values close to 1 correspond to rough surfaces. For $m \to 0$ the term D converges to Dirac δ-function. Commonly used distributions are for instance the Beckmann distribution ($D = \frac{1}{m^2 \cos^4 \alpha} e^{-(\tan \alpha / m)^2}$), the Blinn distribution or the Gaussian distribution.
- $G(L, V, N)$ is the geometry attenuation term, which expresses the amount of light attenuated after shadowing-out in the surface microfacet structure. Please refer to [3] for the complete formula.

The Torrance-Sparrow model was introduced to computer graphics by Cook and Torrance [3], who added a diffuse term and some minor modifications into the model. It is suitable for modelling wide variety of surfaces, especially metals. It is also successful in predicting phenomena such as off-specular reflection and specular backscattering. The HLSL code of this model can be found in Appendix B.

The Oren-Nayar model represents a generalization of the standard Lambertian reflectance. It is similar to the Torrance-Sparrow model in its assumptions; the only difference between them is that the Oren-Nayar model treats each individual microfacet as diffuse reflector, not as a prefect mirror. This complicates the situation, because it is now necessary to take multiple interreflections between neighbouring microfacets into account. The resulting model from Oren and Nayar is therefore only an approximation of the full solution, although a very good one. Its full formulation can be found in [15].

The model is suitable for modelling rough materials with dominant diffuse component, such as clay, stone or uncoated paper. It is capable of reproducing the effect of diffuse backscattering, characteristic for example for the full Moon. Despite its moderate computational costs, it is rarely used in real-time rendering. The reason for this is that the standard 'N-dot-L' Lambertian reflectance exhibits very similar reflectance behaviour, but is far superior in terms of computational performance.

2.2 Layered Reflectance Models

Due to the frequent occurrence of layered materials in our environment, the search for model capable of rendering such materials has received adequate attention. Kubelka and Munk [12, 10, 11] developed a physical model for modelling subsurface scattering within multiple layers. Hanrahan and Krueger [8] presented a model for subsurface scattering computation in the context of computer graphics, and made it directly usable in a Monte-Carlo renderer. These models are comprehensive, but lack an analytical closed form, and therefore are not suitable for real-time applications.

As for analytical models, Neumann and Neumann [14] proposed a simple model for modelling multiple layers. However, they considered only perfectly smooth, transparent layers without including internal reflection. Kelemen and Szirmay-Kalos [9] presented a composite BRDF derived from the Cook-Torrance model. Their model does not explicitly consider surface layers and therefore does not account for absorption and internal reflections, but thanks to the tight coupling between diffuse and specular BRDF components, their model estimates the appearance of single-layered surfaces fairly well.

Finally, the model of Weidlich and Wilkie [21] accounts for both internal reflection and absorption and supports unlimited number of layers. Each layer can have any arbitrary BRDF; the only requirement on these BRDFs is that for all layers except the lowest a transmission component, which allows the light enter the layer underneath, must exist (for example the Cook-Torrance model does contain a transmissive component, while the Oren-Nayar model does not). The only limitation is that the model does not support scattering within the layers. Since this model forms the starting point for our work, we will briefly review it now.

It is worth mentioning that the model relies on the assumption of layers, which are thin in comparison with the surface geometric features. This is not uncommon; in fact all previously mentioned models rely on this. This allows to assume that all incident and refracted rays meet at a single point at every layer (which is very convenient, as it allows purely local evaluation of the model, without including surface spatial position into the model). Fortunately, this condition, along with the assumption of nonpresent subsurface scattering, still hold true for wide variety of materials.

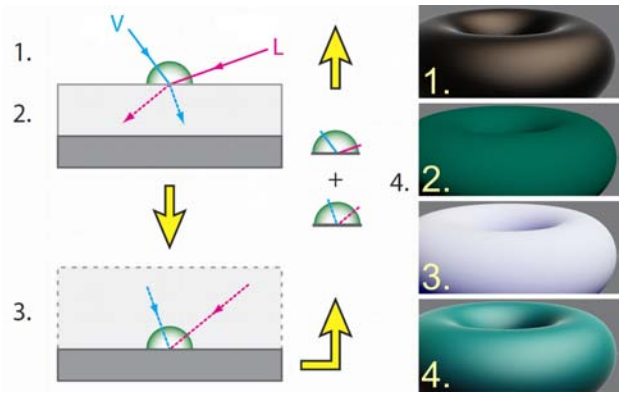


Figure 1: The recursive evaluation scheme. Each stage is also shown graphically. Scheme used with permission.

The evaluation of the model is carried out in a recursive manner and can be described in four steps for two given layers i and $i+1$ (see also Equation 2 and Figure 1):

1. The BRDF of the upper layer f_{r_i} is evaluated for given light and view directions L and V. This also produces the trans-

mittance coefficient $T_{i\to i+1} = 1 - F_i$. Two refracted directions L' and V' are calculated, according to Snell's law for given refractive indices n_{i-1} and n_i ($n_0 \approx 1$ if the object is in the air).

2. The refracted light is attenuated by the absorption term a_i.
3. The BRDF of the lower layer $f_{r_{i+1}}$ is evaluated for L' and V'. If the layer $i+1$ is not the lowest one, we recursively continue from Step 1 ($f_{r_{i+1}} \equiv f_r^{(i+1)}$).
4. On return from the recursion, the light coming from the lower layer is attenuated by $T_{i+1\to i}$ and subjected to possible total internal reflection t_i. The contributions from both layers are added together.

This can be expressed in the form of a recurrent equation for the composite BRDF at layer i as:

$$f_r^{(i)} = f_{r_i}(L,V) + T_{i\to i+1} \cdot f_{r_{i+1}}(L',V') \cdot a_i \cdot t_i \qquad (2)$$

- a_i is the attenuation term according to Bouguer-Lambert-Beer law. The portion of absorbed light depends on the material-specific wavelength-dependent absorption coefficient σ and the distance the light travels in a particular layer:

$$a_i = e^{-\sigma l_i} \qquad l_i = d_i \cdot \left(\frac{1}{N\cdot L'} + \frac{1}{N\cdot V'}\right) \qquad (3)$$

  where d_i is the thickness of the layer i.

- t_i is the internal reflection term. It compensates for the energy lost during the potential total internal reflection of light when crossing an inter-layer boundary from denser into a less dense medium on its way upwards. It is defined as

$$t_i = (1 - G_i) + T_{i+1\to i} \cdot G_i \qquad (4)$$

  where G_i is the Torrance-Sparrow geometry attenuation term.

The final value of the entire model is simply obtained as $f_r = f_r^{(1)}$. For the detailed discussion of the model, please refer to the original paper [21].

2.3 Specialized Material Models

Specialized material models are utilized when the available general BRDFs cannot reproduce the desired material's appearance well enough. As a consequence there is a large variety of such models, each aiming to simulate a single particular effect. Therefore we will list only those few which are relevant for us here; for a comprehensive overview of these models, please refer to [5].

Modelling of car paint is a subject of intensive research, since it is important for virtual prototyping in the automotive industry. Takagi et al. [18, 19] developed techniques for both acquisition and rendering of car paints, that are directly applicable in the industry. Ershov et al. [6] developed an interactive model for pearlescent car paint rendering by simulating scattering between virtual thin sublayers. Recently, Rump et al. [17] introduced a realistic hybrid model for metallic car paint rendering that combines acquired BTF data and classical BRDFs, such as Cook-Torrance model.

As for modelling of metallic patinas, Dorsey and Hanrahan [4] presented a method for simulating growth of patinas on metallic objects by considering multiple layers of material and applying eroding operators on them. For rendering they used the Kubelka-Munk theory with three layers for copper substrate, tarnish and the patina itself. Yao-Xun and Zen-Chung [22] simulated patina growth on objects buried underground, using L-systems. However, these works are focused on patina development simulation and not on rendering.

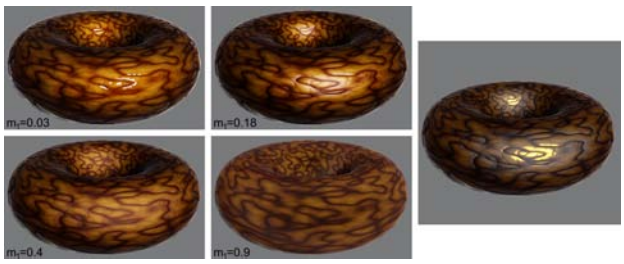


Figure 2: Variation of the upper layer's roughness influences shape of the reflection from the lower layer with $m_2 = 0.3$. The image on the right shows a torus with $m_1 = 0.4$ and $m_2 = 0.05$ without the correction.

3 Layered Materials in Real-Time

The basic version of the real-time layered model adaptation is a relatively straightforward implementation of the evaluation scheme presented in Section 2.2. Algorithm 1 shows a Cg fragment shader for the model with two layers, one light source and an environment reflection from the upper layer. Both layers use the Torrance-Sparrow BRDF, plus a diffuse component for the lower layer, and are normal-mapped.

The main difference is of course the lack of capacity to explicitly cast sampling rays. This has several implications. First, we must strictly stick to the evaluation of the local model with given L and V directions (or L' and V' for the lower layer, respectively). Also the environment reflection must take into account the upper layer roughness, and without sampling this can be achieved only by providing an adequately blurred environment map for the texture lookup at Line 29. Otherwise an inconsistency between the environment reflection sharpness and the specular highlight shape will occur.

Another consequence of the inability to actually sample the BRDF is that a discrepancy between the roughness of the layers might cause an incorrect appearance of the surface, specifically when $m_1 > m_2$. Figure 2 depicts the problem. This cannot happen in Monte-Carlo rendering, because the light gets properly blurred during the refraction on the upper layer. The solution here is to clamp the value of m_2 to the value of m_1 (see Line 21), so that the lower layer's roughness is always greater or equal to the one of the upper layer.

Although the shader we show uses only two layers, there is no obstacle to using more layers (except performance considerations), thanks to the recurrent character of Equation 2. Adding a layer would mean calculation of a new pair of refracted vectors L'' and V'', terms a_2 and t_2 for the new layer and of course of its own BRDF. Another means of enhancing the model's visual richness would be for example adding a thickness map for the upper layer (as in Figure 2), or using a roughness map to vary m across the surface.

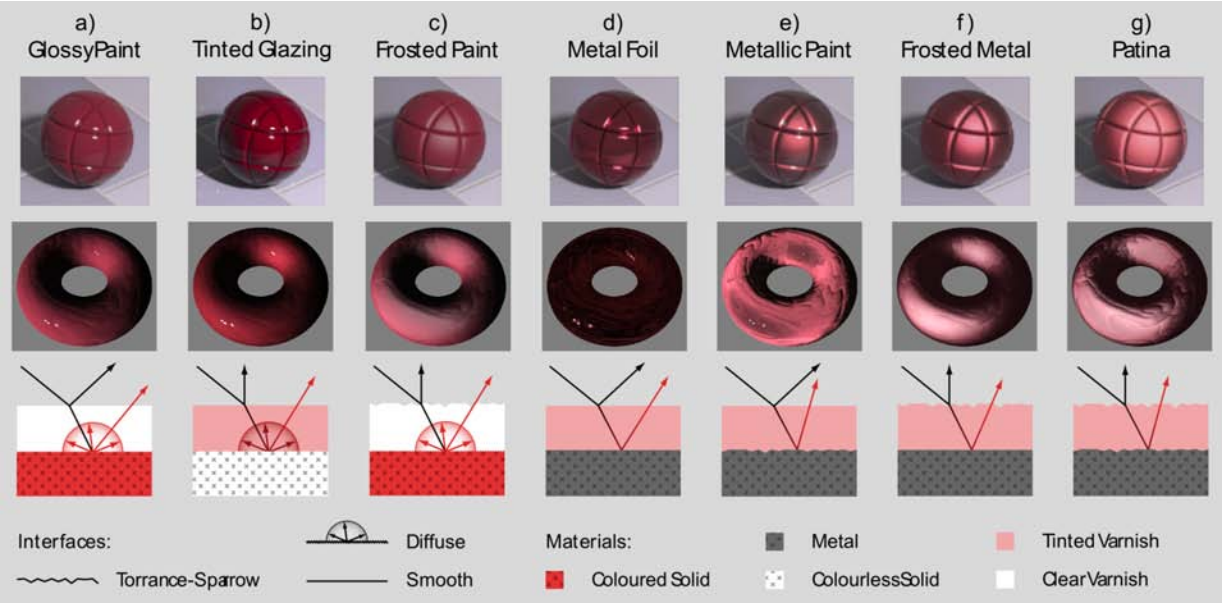


Figure 3: Examples of surfaces that can be generated by the layered model. Top row: Results from the original paper. Bottom row: Images generated by our real-time implementation, with parameters adjusted to the best visual match with their corresponding Monte-Carlo versions. Note that the only qualitative difference is the absence of global illumination. The original image used with permission.

3.1 Optimizations

Undoubtedly the largest performance impact is caused by evaluation of the Torrance-Sparrow model. We will therefore try to speed up its evaluation, what practically means optimizing evaluation of F, D and G terms. We will show that it is possible to precompute F and D.

Fresnel term $F(\theta, n, \kappa)$: The Fresnel term evaluation is the most expensive operation in the model, and is called three times in the basic version of the layered model. The value of F depends on the angle θ between two considered vectors and on the material index of refraction (n,κ). This yields 7 degrees of freedom (for RGB colour components), making a naive precomputation impossible. Of course, it would be possible to separate the R, G and B components of (n,κ) and create three 3D tables, one for each colour component. This is however not the best solution, because it would increase the number of lookups needed to evaluate F from one to three. The solution lies in the fact that for a given material, the value of (n,κ) is fixed, which allows creation of a 1D table parameterized only by the incident angle θ. Such 1D table would contain all possible values of F for that particular material and would be very compact — since the gradient of F in respect to θ is low, the resolution of this table can be small, for example 128 samples. For R8G8B8 texture (which is sufficient, since the value of F is always $\in \langle 0,1 \rangle$) this means the size of only 384 bytes. This allows for creation of a 2D texture atlas for hundreds of materials in the scene with the size in the order of tens of kB.

If precomputation is not desired for some reason, F can be approximated. Lazányi and Szirmay-Kalos [13] presented an accurate and inexpensive approximation of the full Fresnel term, which deviates from the full formula in 5% at most.

Figure 4: Zoom on a sharp specular highlight ($m_1 = 0.01$) using tabulated D term with linear mapping (top), nonlinear mapping (middle) and the full evaluation (bottom). The right image shows a highlight when MIP-mapping is enabled on the texture which contains D.

Distribution term $D(\alpha, m)$: Dimensionality is not an issue here, since both α and m are scalars, so a 2D table can hold the entire distribution term. The complication here is the convergence of D towards the Dirac δ-function when $m \to 0$, independent of which distribution is used. This implies that for very small values of m and $\alpha \to 0$, the large gradient of D cannot properly be reproduced, even if a large sampling rate is used for α (see Figure 4). To remedy this problem, a non-linear mapping must be used for α — on the coordinate $u \in \langle 0,1 \rangle$ the texture holds a value of D, which would be stored in the linearly-mapped texture on the coordinate u^x. We use $x = 8$, since u^8 can be computed in 3 multiplications. But still, even with linear mapping, this issue starts to be apparent just for very smooth surfaces (with $m < 0.02$).

It is also better to disable MIP-mapping for the texture containing D. The reason for this is that the higher MIP levels will blend together adjacent values of the texture (which correspond the different values of m), resulting in an incorrect size and jaggedness of the specular highlight, when viewed from distance. As for the texture resolution, we use a single-channel 16b

```
    Data: vertex shader output structure IN, uniform variables m1, m2, d
          (float) and n1, κ1, n2, κ2, σ, lightCol (float3), texture samplers
    Result: fragment colour
 1  begin
        // calculate involved vectors...
 2      float3 N = normalize(2 * tex2D(normalMap, IN.UV).xyz - 1);
 3      float3 L = normalize(IN.lightDir);
 4      float3 V = normalize(IN.eyePos - IN.fragmentPos);
 5      float3 H = normalize(V + L);
 6      float3 R = reflect(-V, N);
 7      float3 L' = -refract(L, N, 1/n1);
 8      float3 V' = -refract(V, N, 1/n1);
 9      float3 H' = normalize(V' + L');
        // ...and their dot products
10      float NdotL = dot(N, L);
11      float NdotH = dot(N, H);
12      float NdotV = dot(N, V);
13      float VdotH = dot(V, H);
14      float NdotL' = dot(N, L');
15      float NdotH' = dot(N, H');
16      float NdotV' = dot(N, V');
17      float V'dotH' = dot(V', H');
        // BRDFs for both layers
18      float3 F1, F2;
19      float G1, G2;
        // F and G are 'out' parameters
20      float3 f1 = TorranceSparrow(NdotL, NdotV, NdotH, VdotH, n1,
        κ1, m1, F1, G1);
21      float3 f2 = TorranceSparrow(NdotL', NdotV', NdotH', V'dotH',
        n2, κ2, max(m2, m1), F2, G2);
        // diffuse contribution of lower layer
22      f2 += (1 - F2) * max(NdotL, 0) * tex2D(diffuseMap, IN.UV);
        // internal reflection term
23      float3 T12 = 1 - F1;
24      float3 T21 = T12;
25      float3 t = (1 - G1) + T21 * G1;
        // attenuation term
26      float l = d * (1/NdotL' + 1/NdotV');
27      float3 a = exp(-sigma * l);
        // environment reflection for upper layer
28      float3 F1env = FresnelTermNP(NdotV, n1, κ1);
29      float3 envCol = F1env * texCUBE(environmentMap, R);
        // final summation
30      float3 fr = lightCol * (f1 + T12 * f2 * a * t);
31      return float4(fr + envCol, 1);
32  end
```

Algorithm 1: The layered model shader code.

floating-point texture with 512 samples for α and 512 samples for m, resulting in the size of 0.5MB.

Geometry term $G(L,V,N)$: The precomputation of the geometry term is not necessary, as most of the involved dot products have to be calculated anyway, leaving only a few multiplications and two divisions to be evaluated. However, should such need arise, Kelemen and Szirmay-Kalos [9] showed a way to exclude the evaluation of G from the Torrance-Sparrow model.

Of course, a completely different way of speeding up the computation can be taken — instead of using the Torrance-Sparrow model, one can use a simpler BRDF for the layers. Even the Phong model can be used, with the transmission term T derived from the amount of reflected light or by explicitly evaluating the Fresnel term. However, usage of the Phong model decreases the richness of appearance that can be achieved with physically plausible models.

A comparison between the full and precomputed model versions can be seen on Figure 5. A comparison between the results of the real-time version of the model and the original Monte-Carlo implementation of Weidlich and Wilkie can be seen in Figure 3. Figure 10 shows more examples of the model usage.

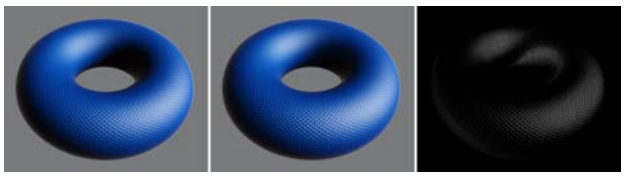


Figure 5: Visual comparison of the full evaluation (left) and using precomputed tables for F and D (middle). The right image shows a magnified difference between the two ($|full - precomputed|$); the largest error is less than 6%.

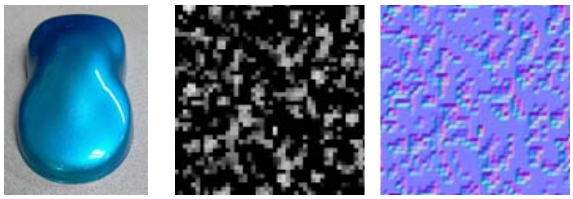


Figure 6: Left: Sparkling effect on an object coated with metallic paint. Right: An example of a texture representing the metallic flakes heightfield and the corresponding normal map.

4 Specialized Materials Modelling

In this section we will present two specialized models for modelling the appearance of metallic car paint and metallic patina. Both are straightforward modifications of the model discussed previously.

4.1 Metallic Paint

Standard solid car paints usually consist of an opaque pigment layer sprayed over base substrate. Metallic paints use translucent pigments which in addition contain very small metallic flakes and can optionally be covered with a clear coating. This structure is responsible for two characteristic appearance features of metallic car paints — an overly metallic look (naturally caused by the material the flakes are made of, e.g. chromium) and sparkling effect, which is especially visible under direct sunlight. This sparkling effect is caused by the fact that the flakes spread in the medium are almost randomly oriented and reflect the incoming light to different directions (see figure 6 for illustration).

To model a surface with such structure it is natural to use the layered model. We use two layers, a lower 'substrate' layer made of chromium and a tinted 'coating' layer. The sparkling effect is achieved by perturbing the surface normal with the texture shown in Figure 6, but only for the lower layer. This normal map is tiled many times across the surface, so that the individual flakes are not visible.

Two problems arise from this approach. The first problem is the consequence of the fact that the flakes are smaller than the size of a pixel. This is modelled by the aforementioned tiling of the perturbing normal map. By doing this, however, a MIP map of the texture is used instead of the full texture to fetch the value of the perturbed normal. This effectively smoothes out all

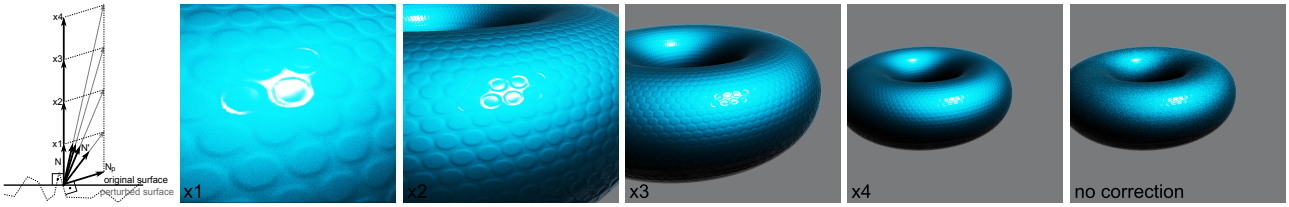


Figure 7: Left: The scheme depicting the scaling of the perturbed normal. Middle: Four distances from the object corresponding to the scheme. Right: The object without the scaling correction.

details provided by the flakes normal map. To overcome this, all filtering must be disabled for the normal map. This produces the effect of noisy sparkling, which dynamically changes as the observer moves.

The second problem arises from the solution to the first one. Minification filtering and MIP mapping ensure that high-frequency features on the texture are filtered so aliasing does not occur. By disabling these, the noisy sparkling is apparent even from distance. In reality this cannot happen, because as the observed object gets further from the eye, more flakes are projected onto the same area on the retina, effectively averaging and smoothing the perceived image. This can be solved by scaling the original surface normal by the distance of the observer from the fragment and add this scaled normal to the perturbed normal (see Figure 7). This effectively decreases the influence of the perturbed normal, making the surface look smooth from distance. In code, this can be written as:

```
float distance = length(IN.eyePos - IN.fragmentPos);
    // N currently contains the normal-mapped normal
    // also remember we are in tangent space
N += max(distance, 1) * float3(0, 0, 1) +
  (2 * tex2D(flakesNormalMap, 1000 * IN.UV).xyz - 1);
N = normalize(N);
```

Multiplication of the UV coordinates by a large number tiles the normal map. This block of code is to be inserted between Lines 13 and 14 in the Algorithm 1 (to influence only the angles between the refracted rays).

4.2 Patina

Patination is a chemical oxidation process which occurs on metals. It changes the chemical composition near the surface of the material, often resulting in a layer of substance with different optical properties than the original material. Unlike rusting, patination does not destructively erode the metal; instead, it forms a solid protective layer atop of the metal substrate, which then stays stable. Patination occurs on many common metals and alloys, for example on copper, brass, aluminium, tin and even on silver. We chose to model copper patina, because of its distinct appearance.

Copper patinas often have complex chemical composition, which tend to differ with the atmospheric conditions the copper object is exposed to. The involved substances are cuprite Cu_2O, antlerite, brochantite and possibly others. Since the physical constants for these are not widely available, we use for the patina the IOR of cuprite and an empirically matched light-green absorption spectrum.

An important thing is to have control over the patina growth, i.e. to determine places where patina is already developed and

Figure 9: Left: A grayscale map that controls the patina development. Right: Blurry (`transition` $= 0.9$) and sharp (`transition` $= 0.2$) boundaries of the developed patina regions.

where not yet. Naturally, this changes over time. To model the development we use a single grayscale texture (see Figure 9) and two parameters: $transition \in \langle 0,1 \rangle$ and $extent \in \langle -transition, 1+transition \rangle$. The first one controls the sharpness of the patina regions and the second one controls the extent of patina development (the larger is the *extent* parameter, the more of the surface is covered with patina). As for the texture, the darker the value, the earlier will the patina develop on that particular position. The texture can be derived from the surface curvature (as in our case, since the patina develops earlier in places like cracks and wrinkles) or can be an output from an actual weathering simulation. The code for this looks as follows:

```
float patinaValue = tex2D(patinaMap, IN.UV).r;
float extentFactor = 1-smoothstep(extent - transition,
                                  extent + transition,
                                  patinaValue);
```

The variable $extentFactor \in \langle 0,1 \rangle$ is then used through the entire shader to control the amount of patina.

It is necessary to realize that on the places where the patina is not developed yet, only the reflection from the lower layer has to be taken into account. The development of patina also changes some properties of the lower layer, for example its roughness. Figure 8 demonstrates this process. Therefore the model has to be evaluated for both cases (with and without the upper patina layer) and the *extentFactor* variable should be used to linearly interpolate between them. The *extentFactor* should be also used to interpolate between the model parameters influenced by the patina growth, namely the aforementioned roughness parameter m_2 and the IOR value (n, κ) used to calculate the environment reflection (Line 28 in Algorithm 1).

Unfortunately, the substance that forms the patina layer does not conform to the assumption that no light is scattered within the layer. Quite the contrary — it is the scattering that makes the patina appear primarily as a diffuse reflector. To avoid subsurface

Figure 8: Patina development 'over time' by changing the *extent* parameter value (from left to right -0.65, -0.05, 0.15, 0.45, 0.75 and 1.65; *transition* $= 0.65$).

scattering computation we approximate this effect by adding a diffuse component to the BRDF of the patina layer, that is:

```
float3 patinaDiffuse = (1 - sigma)
                * (1 - F_1) * max(NdotL, 0);
f_1 += patinaDiffuse;
```

We modulate the diffuse component by the remainder of the absorption spectrum σ to strengthen the characteristic colour of the patina. To further improve the appearance of patina, we control its thickness with an additional texture (as proposed in Section 3) and modulate the base copper layer with a dark-orange tone to mimic the appearance of tarnish.

5 Results and Conclusions

We implemented the basic model and the derived specialized models in HLSL, using NVIDIA FX Composer 2.5 for development and measurements. We test the basic two-layer model in two versions, using the full evaluation and using precomputed tables for F and D, against a single-layer Phong shader with similar features (normal mapping, environment reflection). We do not measure the two specialized models, as these are simple modifications of the basic model and do not add significant computational overhead. The measurements used GeForce 8800 GTX (G80) as a reference GPU. Table 1 summarizes the measurements.

Model	GPU cycles	MPix/s
Layered (full)	436	348
Layered (precomp.)	236	757
Phong	104	1648

Table 1: Performance comparison of the layered model and standard Phong model. The measured quantities are the number of G80 GPU cycles used for model's evaluation and the corresponding pixel throughput.

So, the discussed layered model is still roughly 2.3 times slower than the Phong model. This is expected however, since we are evaluating two BRDFs instead of one, and each of them being far more complex than the Phong model.

Conclusions Although the discussed layered model is slower than the commonly used reflectance models in real-time 3D applications, it provides superior appearance reproduction of a wide variety of surfaces, and is physically plausible. Moreover, it is very likely that the performance impact of using this model in a real-time application would be only a few percent, because:

- It is not necessary to use the model on all objects in a scene, but only on objects in the user's primary attention (e.g. cars in a racing game).
- The performance of any renderer is not given solely by the performance of the used reflectance models, but is mainly a consequence of many other tasks the renderer have to perform.

This indicates that the model is a viable alternative for use in today's real-time 3D applications, including games.

To conclude the paper; we have presented a real-time implementation of the physically-based layered BRDF introduced by Weidlich and Wilkie [21]. We have then shown how to optimize this model to be useful in real-time 3D applications. Furthermore, we have explored the capabilities of this model by deriving two specialized models for modelling the appearance of metallic car paint and metallic patina from it. The FX Composer project containing all discussed shaders will be made available on the author's webpage.

6 Acknowledgements

I would like to thank Alexander Wilkie for his insights and advice regarding this work.

References

[1] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of SIGGRAPH '77*, pages 192–198, 1977.

[2] M. F. Born and E. Wolf. *Principles of Optics*. Cambridge University Press, 7th edition, 1999.

[3] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. In *Radiometry*, pages 42–59, 1992.

[4] J. Dorsey and P. Hanrahan. Modeling and rendering of metallic patinas. In *Proceedings of SIGGRAPH '96*, pages 387–396, 1996.

[5] J. Dorsey, H. Rushmeier, and F. Sillion. *Digital Modeling of Material Appearance*. Morgan Kaufmann Publishers, 2008.

[6] S. Ershov, K. Kolchin, and K. Myszkowski. Rendering pearlescent appearance based on paint-composition modelling. *Comput. Graph. Forum*, 20(3), 2001.

[7] A. S. Glassner. *Principles of Digital Image Synthesis Volume Two*. Morgan Kaufmann Publishers, 1995.

Figure 10: Examples of the model usage. Left: Car paint with strong sparkling effect. Middle: Varying upper layer thickness (orange and purple lacquers used). Right: Concrete ball coated in a blue transparent varnish. The lower concrete layer uses the Oren-Nayar BRDF.

[8] P. Hanrahan and W. Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of SIGGRAPH '93*, pages 165–174, 1993.

[9] C. Kelemen and L. Szirmay-Kalos. A microfacet based coupled specular-matte BRDF model with importance sampling. In *Eurographics Short Presentations*, pages 25–34, 2001.

[10] P. Kubelka. New contributions to the optics of intensely light-scattering materials. Part I. *J. Opt. Soc. Am.*, 38(5):448–448, 1948.

[11] P. Kubelka. New contributions to the optics of intensely light-scattering materials. Part II: Nonhomogeneous layers. *J. Opt. Soc. Am.*, 44(4):330–334, 1954.

[12] P. Kubelka and F. Munk. Ein beitrag zur optik der farbenstriche. In *Z. tech. Physik 12*, pages 593–601, 1931.

[13] I. Lazányi and L. Szirmay-Kalos. Fresnel term approximations for metals. In *WSCG 2005 Short Communications Proceedings*, 2005.

[14] L. Neumann and A. Neumann. Photosimulation: Interreflection with arbitrary reflection models and illumination. *Comput. Graph. Forum*, 8(1):21–34, 1989.

[15] M. Oren and S. K. Nayar. Generalization of Lambert's reflectance model. In *Proceedings of SIGGRAPH '94*, pages 239–246, 1994.

[16] B. T. Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[17] M. Rump, G. Müller, R. Sarlette, D. Koch, and R. Klein. Photo-realistic rendering of metallic car paint from image-based measurements. *Comput. Graph. Forum*, 27(2), 2008.

[18] A. Takagi, H. Takaoka, T. Oshima, and Y. Ogata. Accurate rendering technique based on colorimetric conception. In *Proceedings of SIGGRAPH '90*, pages 263–272, 1990.

[19] A. Takagi, A. Watanabe, and G. Baba. Prediction of spectral reflectance factor distribution of automotive paint finishes. *Color Research and Application*, 30(4), 2005.

[20] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. In *Radiometry*, pages 32–41, 1992.

[21] A. Weidlich and A. Wilkie. Arbitrarily layered micro-facet surfaces. In *GRAPHITE 2007*, pages 171–178, 2007.

[22] Ch. Yao-Xun and S. Zen-Chung. Physically-based patination for underground objects. *Comput. Graph. Forum*, 19(3), 2000.

A Selected coefficients

The following table lists a few selected indices of refraction for materials that have been used in the paper (in the models for metallic car paint and metallic patina rendering).

Material\λ	**r**[690nm]	**g**[550nm]	**b**[450nm]
Copper (Cu)			
n	0.213	1.04	1.17
κ	4.05	2.59	2.36
Chromium (Cr)			
n	3.84	3.18	1.99
κ	4.37	4.41	4.22
Cuprite (Cu_2O)			
n	2.83	3.10	3.06
κ	0.083	0.19	0.6

B Torrance-Sparrow model

```
float3 TorranceSparrow(float NdotL, float NdotV,
                       float NdotH, float VdotH,
                       float3 n, float3 k, float m,
                       out float3 F, out float G)
{
      //D term - Beckmann distribution
    float D;
    float tg = sqrt(1 - NdotH * NdotH) / NdotH;
    D = 1 / (m * m * NdotH * NdotH * NdotH * NdotH)
             * exp(-(tg/m) * (tg/m));

      //F term - Lazanyi-Szirmay-Kalos approximation
    float q = 1 - VdotH;
    F = ((n - 1)*(n - 1) + 4 * n * q*q*q*q*q + k*k)
        / ((n + 1)*(n + 1) + k*k);

      //G term
    G = min(1, min(NdotV * (2 * NdotH) / VdotH,
                   NdotL * (2 * NdotH) / VdotH));

      //entire model
    return F * D * G / (4 * NdotV);
}
```

Computer Vision

Comparison of face recognition algorithms in terms of the learning set selection

Simon Gangl*
Domen Mongus†
Supervised by: Borut Žalik‡

Laboratory for **Ge**ometric **M**odelling and **M**ultimedia **A**lgorithms
Faculty of Electrical Engineering and Computer Science
University of Maribor / Slovenia

Abstract

A suitable selection of facial features is of key importance for the successfulness of face recognition algorithms. Because a straightforward selection of them does usually not ensure sufficient reliability, statistical tools are often used for feature extraction. In this paper the influence of the selected set of learning samples on the efficiency of face recognition algorithms is observed. For this purpose, three of the most often used algorithms are presented in detail. The feature description based on the Gabor wavelet transformation is presented first. In this approach features are selected based on human physiognomy basis and formed to feature graphs, where the actual recognition is performed by graph matching. On the other hand, principal component analysis (PCA) is a statistical tool for identifying patterns in data by reducing its dimensionality. That way, key features for face recognition are extracted to a comparable form. Meanwhile, linear discriminant analysis (LDA) allows for face recognition by establishing the borders between classes in multidimensional data. To ensure equal conditions for those algorithms, a method for image normalization is presented also. By the results it is shown, that the statistical approaches are significantly more reliable yet at the same time strongly dependant on the learning set selection. Even if no significant influence of the learning set on the Gabor wavelets based method can be observed, its successfulness is clearly below those of PCA and LDA.

Keywords: Face recognition, PCA, LDA, Gabor wavelets, Learning set selection

1 Introduction

Although various methods for face recognition have been developed, it remains an important field of research. One of the main reasons certainly lies in high market demands for secure systems based on biometrical identification. Face recognition is recognized as one of the more elegant approaches, since it is user-friendly as well as cost-efficient. At the same time, findings in face recognition research are often applied to industrial projects for the purpose of pattern recognition in general [1]. Whatever the purpose may be, the efficiency is strongly dependent on the detected features and the quality of their representation in the model base [2]. Although many approaches are known for this task [2, 3], features are usually assembled as components of a feature vector [2, 5, 6, 7]. In such cases, each component of a vector caries important information, which is the basis for distinguishing between faces. In a most simple case, features can describe the colour of the human eye, the colour of the skin or the shape of the face, but unfortunately such simple features are usually not sufficient enough. Therefore, statistical techniques are often used to determine adequate features. When the feature extraction is based on statistical attributes of the selected face population, then the final set of features may be very dependent on the subset of faces that were used in the learning process. Because of that it makes sense to study the possible influence on the accuracy of the face recognition algorithms.

In this paper we present a study of the influence of the training set selection on the face recognition accuracy. For this purpose the training set dependency of three algorithms was analyzed: feature graphs based on a wavelet transform, principle component analysis (PCA), and linear discriminant analysis (LDA). The efficiency of recognition techniques was compared in terms of dependency on the learning set of faces.

A detailed review of the image normalization procedure, which ensures robust detection of features and provides equal conditions for testing the efficiency of methods, is given in Section 2. This is followed by a detailed presentation of the used face detection techniques (Section 3). Results, obtained using these procedures are presented in Section 4. The most important conclusions are emphasized in Section 5.

*simon.gangl@uni-mb.si
†domen.mongus@uni-mb.si
‡zalik@uni-mb.si

2 Input image normalization

Main issues that need to be considered when dealing with computer based detection and recognition systems are related to unequal light distribution, camera position, image quality, and image resolution. The capabilities of such systems can be reduced dramatically by these parameters. Therefore, the elimination of these factors needs to be accomplished before the recognition is performed. This process is called image normalization [5, 6] and is, in our case, achieved in four steps:

Step 1 (face detection): The detection of the face is usually the first step of the image normalization process. A neural network is used for this purpose. In the training process the neural network was trained to detect the presence of a face in an image with resolution of 128x128 pixels. Such a neural network is capable of detecting faces only in images with the same resolution. Because the input images are of arbitrary size, face detection cannot be performed directly. Therefore a sliding window is defined. By testing the sliding window region for the presence of a face at each position, faces can be detected. That way all faces in the input image, located at one of the regions sized 128x128 pixels, can be found. Nevertheless, the faces in input images are usually much larger and are not detected at this step. Therefore the input image size has to be reduced multiple times and scanning for faces repeated at each iteration. For this task a sufficient scaling factor has to be chosen, which is a trade-off between execution speed and reliability of detection. In our case a scale factor of 0.9 is used that assures us with 100% face detection ratio on the FERET database [4].

Step 2 (histogram equalization): According to the detected face region, the image is then cropped and thus the background is removed. However, noise, caused by illumination, may still present a disturbing influence. To increase robustness of the face recognition process against that, histogram equalization is performed on the cropped image.

Step 3 (eye detection and rotation of the image): Eye detection is performed during image normalization to increase robustness of the following steps against camera rotation (or rotation of the head) and thus ensures that all faces appear in horizontal position. Similar to the face detection, the detection of eyes is performed with a neural network. The middle points of the eyes are then used to calculate the sufficient angle of rotation θ. The rotation of the image is formally defined by the following equation:

$$x_1' = cos(\theta)\cdot(x_1 - x_0) - sin(\theta)\cdot(y_1 - y_0) + x_0\,, \qquad (1)$$
$$y_1' = sin(\theta)\cdot(x_1 - x_0) - cos(\theta)\cdot(y_1 - y_0) + y_0\,,$$

where (x_0, y_0) is the centre point of the rotation; in our case this is the middle point between the eyes, (x_1, y_1) is the pixel that is transformed at the given step, and (x_1', y_1') are the transformed coordinates of the pixel.

Step 4 (scaling of the image): To achieve the best possible matching among the normalized images, they are scaled so that the centre points of the eyes are located at the same positions on the normalized images. The scaling factor is defined as the ratio between the desired and observed between eye distance.

When scaling the image is completed, an additional mask is applied to it, so the remaining background factors, like hair for example, are eliminated. Examples of normalized images can be seen in Figure 1.

Figure 1: Normalized images, which are the input to face recognition techniques.

3 Face recognition techniques

The process of image normalization leads us intuitively to the possibility of face recognition by feature graph matching. This is the first presented approach, where features for face representation are selected on human physiognomy basis and represented using a wavelet transformation with Gabor wavelets. In the continuation, two more often used techniques for face recognition are presented also. Both are based on a linear transformation of the image to a feature subspace. These techniques are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

3.1 Feature graph matching based on Gabor wavelet transformation

Because of the image normalization process, the introduction of face recognition according to feature graph matching is relatively straightforward. In our case a modified approach presented in [7] is used for this purpose. The presented approach introduces a vector of wavelet coefficients (jet), which carries the facial features at a given key point. The components of such a jet describe the response to a Gabor wavelet transformation at a given key point. Since the key points are at fixed positions, the structures of the graphs are equal and thus the graph matching can actually be performed only by comparing the jets, using the given metric.

The Gabor wavelet transform is employed here because it is robust against variations in illumination and small changes in phase [7]. In our case 40 different wavelets (5 different frequencies at 8 different orientations) are used. Figure 2 shows the construction of such jets, and their formation to a feature graph.

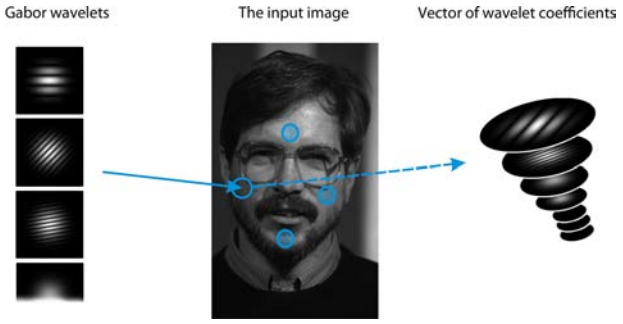


Figure 2: Construction of a vector of wavelet coefficients, where the convolution with the Gabor wavelets is performed at a given point.

The basic difference between our approach and the approach, presented in [7], is that in our case no complete adaptation of the feature graph is needed, since the input images have already been normalized. This way the procedure is much more time efficient, but it also becomes much less flexible. Some important information regarding distances between features is lost, making the recognition less reliable (Section 6).

The facial features are chosen with regard to facial physiognomy [8], where four points, which are important for human facial recognition yet not subject to quick evolution, are chosen. These points are selected at the left and right cheek, on the forehead, and above the chin. At each of the selected points the vectors of wavelet coefficients can now be obtained by calculating the responses to all of the 40 Gabor wavelets. Because even small changes in position can cause a phase shift in the response [7], the wavelet transformation is calculated actually in a 7x7 adjacency of the selected key points. Thus a single face is presented in the model base with 49 feature graphs.

In the process of recognition the jets can now be computed only at the selected key points of the test image. The model base is then searched for the best match to the resulting graph, where the distance between jets is measured with the L^1 or Manhattan metric, defined with the

$$d(\boldsymbol{p},\boldsymbol{q}) = \|\boldsymbol{p}-\boldsymbol{q}\|_1 = \sum_{i=1}^{n} p_i - q_i\,, \tag{2}$$

where $d(\boldsymbol{p}, \boldsymbol{q})$ is the distance between two vectors $\boldsymbol{p}$ and $\boldsymbol{q}$.

3.2 PCA

PCA is a statistical tool for identifying patterns in data. It allows a representation of various sets of data in a way, where similarities between samples are emphasized. Because it is difficult to search for patterns in multidimensional data, PCA is an important tool for data analysis. Nowadays, PCA is present also as one of the most popular approaches for recognition of faces [9, 10, 11] and patterns in general [12, 13]. The implementation of PCA for face recognition can be described in six steps:

Step 1: The inputs of the process are normalized facial images, from which a model database is built. The images are transformed to vectors by dividing them to rows (or columns) which are placed one after another (in our case the images are of dimension 256 x 256, thus each vector has 65.536 components). Each image now represents a base vector of a vector space, with as many dimensions as there are input images. These vectors are formed in a matrix, where each vector represents a column, for a clearer representation.

Step 2: The origin of the vector space is then translated to the point (0, 0, ... , 0) by subtracting the average value of each base vector from its components (the average image intensity is subtracted from its pixels).

Step 3: The dimensionality of the vector space is then decreased by expressing the mutual dependency of the base vectors with a covariance matrix:

$$\boldsymbol{C}_{i,j} = \frac{\sum_{i=1}^{N}(\boldsymbol{x}_i - \bar{x_i})(\boldsymbol{x}_j - \bar{x_j})}{(N_{PCA}-1)}\,, \tag{3}$$

where $\boldsymbol{C}_{i,j}$ is the *(i,j)*-th element of the covariance matrix, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the vectors for which the covariance in the given step is calculated, $\bar{x_i}$ and $\bar{x_j}$ are their average values, which are because of step 2 in our case always 0 and N_{PCA} is the dimensionality of the vectors.

Step 4: The eigenvectors and according eigenvalues of the covariance matrix can then be obtained. Because the eigenvectors represent the interdependency of data, they can be interpreted as facial features in which the patterns from the learning set resemble or differ (Figure 3). Although the obtained vector space allows for face recognition, its efficiency can be increased by discarding eigenvectors corresponding to the highest eigenvalues. These vectors are namely under the influence of illumination distribution and do not resemble valid facial information [11]. In our case, the eigenvectors are sorted descending in terms of their eigenvalues and the first two vectors are discarded. The remaining eigenvectors form the vector subspace $\boldsymbol{E}$, and are presented in matrix form, where each vector represents one column.

Figure 3: The eigenvectors, where the influence of illumination in the input images on the vectors with the biggest eigenvalues can be seen (the upper row), and the vectors with smaller eigenvalues, which represent features (the two bottom rows).

Step 5: In vector space $\boldsymbol{E}$ face recognition can be performed. The base of known faces is created by projecting the input images to the vector space $\boldsymbol{E}$, thus expressing them as a linear combination of the eigenvectors, what can be defined by the following equation:

$$\boldsymbol{y}_i = \boldsymbol{E}^T \cdot \boldsymbol{x}_i \,, \tag{4}$$

where $\boldsymbol{y}_i$ is the projection of the input image $\boldsymbol{x}_i$ to the PCA vector subspace $\boldsymbol{E}$ defined by the reduced matrix of eigenvectors of the covariance matrix $\boldsymbol{C}$.

Step 6: In the process of recognition each input image is projected to the vector subspace $\boldsymbol{E}$ and then compared to the vectors in the model base of known faces using the normalised Euclidean, or the Mahalanobis metric, defined by:

$$d(\boldsymbol{y}_i,\boldsymbol{y}_j) = \sqrt{\sum_{n=1}^{N_{PCA}} \frac{(y_{i_n} - y_{j_n})^2}{\sigma_n^2}} \,, \tag{5}$$

where $d(\boldsymbol{y}_i,\boldsymbol{y}_j)$ is the distance between vectors $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$, σ_i is the standard deviation, which is in our case replaced by the eigenvalue corresponding to the i-th eigenvector.

3.3 LDA

Similar to PCA, also LDA can be used for data classification. LDA is based on maximizing the between-class variance to within-class variance ratio. The most important difference between PCA and LDA is that PCA minimizes the projection error by emphasising similarities between samples; meanwhile LDA defines the classification boarders. Both methods include a projection of data to a subspace, where classification can be performed more accurately. PCA changes the form and location of the input data, while LDA leaves the input data unchanged [14]. In our case LDA is performed globally on the PCA output vectors and this can be described in five steps:

Step 1: The inputs to the LDA process are vectors already projected to the PCA subspace. Because LDA permits many samples belonging to a single class (each person can be presented by multiple images), an additional component is added that defines the class of the vector.

Step 2: The average of each class separately $(\mu_1, \mu_2, ..., \mu_{NR})$ and the average of all classes μ are then computed. The average of all classes is obtained using the following equation:

$$\mu = \sum_{i=1}^{N_R} p_i \mu_i \,, \tag{6}$$

where p_i is the probability of occurrence of a specific class, and can be computed as straightforward as $p_i = 1/N_R$ for all classes, where N_R is the number of all classes.

Step 3: From the data collection two scatter matrices can now be obtained. The scatter matrix $\boldsymbol{S}_w$ describes the expected covariance within each class $R_j; 1 \leq j \leq N_R$, while the scatter matrix $\boldsymbol{S}_b$ describes the scattering between classes. When many samples of a class $\mathbf{y_i^j}; 1 \leq i \leq M_j$, exist, the matrix $\boldsymbol{S}_b$ can be understood as a description of covariance between the average vectors μ_j of each class. The equations for calculating the two matrices can be written as:

$$\boldsymbol{S}_w = \sum_{j=1}^{N_R} \sum_{i=1}^{M_j} (\boldsymbol{y}_i^j - \mu_j)(\boldsymbol{y}_i^j - \mu_j)^T \,, \tag{7}$$

$$\boldsymbol{S}_b = \sum_{j=1}^{N_R} (\mu_j - \mu)(\mu_j - \mu)^T \,,$$

Step 4: As already mentioned the LDA optimization criterion is defined as the ratio between $\boldsymbol{S}_w$ and $\boldsymbol{S}_b$. Since Fisher LDA is used, the optimization criterion can be written as:

$$J(\boldsymbol{W}) = \frac{\boldsymbol{W}^T \boldsymbol{S}_b \boldsymbol{W}}{\boldsymbol{W}^T \boldsymbol{S}_w \boldsymbol{W}} \,, \tag{8}$$

where the matrix $\boldsymbol{W}$ is obtained by maximizing the value J($\boldsymbol{W}$). Although this can be achieved by several methods, in our case the ratio $det|\boldsymbol{S}_b|/det|\boldsymbol{S}_w|$ is maximized [15]. It has already been shown that, if $\boldsymbol{S}_w$ is a nonsingular matrix, the ratio is maximized, by forming the columns of $\boldsymbol{W}$ from the eigenvectors of $\boldsymbol{S}_w^{-1}\boldsymbol{S}_b$ [16]. Although in real cases $\boldsymbol{S}_w$ is usually nonsingular, its non-singularity can be ensured by using at least two samples of each class [15]. After that, the matrix $\boldsymbol{W}$ is normalized before it is used in the following procedures.

Step 5: $\boldsymbol{W}$ now represents the vector subspace in which, according to the given set of learning patterns, optimal

classification can be performed. Formally the process of recognition in the LDA space can be defined with the following equation:

$$\begin{aligned} \boldsymbol{y}_i &= \boldsymbol{E}^T \cdot \boldsymbol{x}_i \,, \\ \boldsymbol{z}_i &= \boldsymbol{W}^T \cdot \boldsymbol{y}_i \,. \end{aligned} \tag{9}$$

where $\boldsymbol{E}$ is the vector space of PCA, $\boldsymbol{W}$ is the projection matrix of LDA, $\boldsymbol{y}_i$ the image $\boldsymbol{x}_i$ projected to PCA subspace and $\boldsymbol{z}_i$ the projection of $\boldsymbol{y}_i$ to subspace $\boldsymbol{W}$. The patterns are compared using the Euclidean or L^2 metric, defined by the following equation:

$$d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \sqrt{(z_{i_1} - z_{j_1})^2 + \ldots + (z_{i_{N_{LDA}}} - z_{j_{N_{LDA}}})^2}\,, \tag{10}$$

where $d(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the distance between N_{LDA}-dimensional vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$.

4 Measurements and results

Tests were performed on the FERET image database [4]. The FERET image database consists of images of more than one thousand people, taken at different time intervals, with different poses and facial expressions. The presented approaches were tested on three model bases, selected from the FERET base. The first base contains 10 individuals, the second 20 and the third 40 individuals, varying in gender, age, pose, and race. For each individual five images were used for testing, while one images was employed as the learning sample. Because the main interest of our work is the influence of the learning set selection, and not the efficiency of the algorithms, only limited numbers of individuals and only one training image per individual were employed. That way this effect can clearly be studied. Table 1 shows the number of correctly identified samples $\bar{x}$ in percents, while the standard deviation σ describes the class variance from the average efficiency.

Table 1: Successfulness of face recognition with Gabor wavelets, PCA, and LDA.

	Gabor wavelets	PCA	LDA
Base 1: $\bar{x}$	68%	80%	77%
Base 1: σ	0.9940	0.5164	0.4830
Base 2: $\bar{x}$	56%	88%	85%
Base 2: σ	1.0940	0.9987	0.9679
Base 3: $\bar{x}$	56%	82%	79%
Base 3: σ	1.3940	1.1873	1.4118

The first, perhaps a bit surprising, result is that PCA as well as LDA produced better results on base 2 than on base 1, where less testing samples were used. The main reason for this is that both methods can become over-determinate [11, 14], when the learning base contains a smaller number of samples. LDA is especially prone to this effect, since PCA is part of it, and thus produces worse results than PCA alone. Because it is evident that statistical methods of recognition need more learning samples to extract important features, it could be expected that both methods would work even more efficient on base 3. But that is not the case, even an unexpected high decrease in effectiveness can be observed.

To study this effect, several testing sets were employed, created by replacing the learning samples of individuals. By doing that, a high increase of efficiency was noticed when a specific image (see figures 4a and 4b) was not included in the training process. The efficiency of PCA increased to 90% (with standard deviation σ = 1,0671) and that of LDA increased to 87% (with standard deviation σ = 0,9901). In figures 4c and 4d the significant influence of an individual on the entire projection space can be seen clearly. When the specific image of an individual is included in the formation of the projection space, features are accented weakly and are under the influence of illumination distribution (figure 4c). This is evident even on eigenvectors with smaller eigenvalues, although this effect would usually be expected only on some of the eigenvectors with higher eigenvalues (figure 4d).

For the verification of the obtained results, a base of known faces with a higher number of training samples was created. In test base 4 images of 236 persons were included (again only one training image per person was used). The results of our experiments have shown that by applying the same methods as mentioned above, the recognition ratio could be improved by 4%. The relative improvement is this time smaller, which is not entirely unexpected, but the number of additionally recognized images is still a good motivation to observe the influence of the learning set selection also on bases including a higher number of samples.

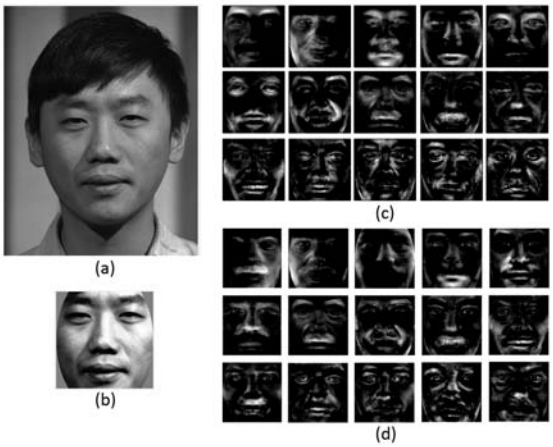


Figure 4: The influence of a learning sample on the eigenvectors, where (a) shows the image of the specific individual, (b) is the normalized form of that image, (c) displays the first 10 eigenvectors when this image was used as a learning sample and (d) are the corresponding eigenvectors when another image of the individual was used.

The selection of the model base has no such particular influence on the recognition method based on Gabor wavelets. The decrease of effectiveness of recognition with increasing number of models in the base is evident, but it is not unexpected. Anyhow, this method produces a relatively unsatisfying result compared to PCA or LDA. The main drawback here is the method for measuring distances between graphs. When using this method, a higher degree of recognition cannot be achieved with simple metrics (like L^1). Because of that, various authors have proposed a use of Gabor wavelet based methods, where the actual comparison is performed with statistically based tools, such as PCA or LDA [17, 18].

Situations, where the tested person is not present in the model base, are often encountered in real-world applications. In such cases the person must be classified as an unknown individual. Because of that, an additional threshold needs to be introduced. If the calculated distance between a sample and its nearest class is greater than the threshold, the given sample is identified as unknown. In our case this threshold is defined as the mean value between the average distance of correctly identified samples and the average distance of a set of unknown samples. For this purpose, additional 100 negative testing samples were included iqnto the previously described test base 3, which contains 200 positive testing samples. The results of this test are shown in Table 2, where the number of correctly identified positive samples is presented as TP (true positives), FN (false negatives) is the number of errors, where a positive sample is recognized as a negative one, FP (false positives) is the number of errors, where a negative sample is recognized as a person from the base and TN (true negatives) represents the number of correctly identified negative samples. The numbers TP and FN sum up to the percentage of correctly identified persons from the test base 3 as shown in Table 1 (thus they represent how many of the previous correctly recognized images are still correctly recognized - TP, and how many are recognized as unknown because of the introduced threshold - FN).

Table 2: Efficiency of face recognition with Gabor wavelets, PCA and LDA, tested on positive and negative samples

	Gabor wavelets	PCA	LDA
TP	50%	80%	76%
TN	81%	96%	94%
FP	19%	4%	6%
FN	6%	2%	3%

From the results shown in Table 2, it can be observed that the introduced threshold does not reduce significantly the efficiency of the presented methods. At the same time, a relatively high percent of negative images is identified. This is most obvious for the PCA and LDA techniques, which again confirms the mentioned fact about the influence of training samples on the efficiency. The distance between a negative sample and its closest class is in most cases significantly greater than the distance from a positive sample to the classification classes. Because of that, there is a higher error rate in recognition of known samples, than of those which are not. Based on the mentioned facts, it can be concluded that the selection of training samples for the creation of the projection subspace is of high importance for the efficiency of PCA and LDA. Consequently, this applies also for methods based on Gabor wavelets, if the actual comparison between features is performed with one of these two techniques.

5 Conclusion

Three approaches to face recognition were presented in this paper; an approach based on the Gabor wavelet transform, PCA, and LDA. Additionally, a method for image normalization, which ensures sufficient conditions for face recognition, was demonstrated. The presented methods were tested on different testing sets with special emphasis on analyzing the influence of learning samples on their efficiency. The first conclusion is that the efficiency of the PCA and LDA techniques improves with an increasing learning set. Because both methods are based on statistical laws, they require a larger set of learning samples that provide high representability. Even further, using PCA or LDA the selection of eigenvectors for the projection subspace formation is of great importance. Some of the eigenvectors associated with the highest eigenvalues namely represent illumination distribution in the learning samples, and it thus makes sense to exclude them. It also makes sense to observe the influence of each learning sample on the efficiency of recognition. It was shown how a single learning sample can noticeably change the projection space and decrease the efficiency of the PCA and LDA techniques. At the same time, it is possible to identify unknown samples reliably by projecting them to the eigenvector subspace. Because these samples were not included in the formation of projection subspace, their features are not emphasized and the distance to the defined classes is noticeably greater. Most of the undesired effects can be omitted using the technique based on the Gabor wavelet transform.

References

[1] C. Liu, H. Wechsler *Comparative Assessment of Independent Component Analysis (ICA) for Face Recognition, pp. 211-216*. Second International Conference on Audio- and Video-based Biometric Person Authentication, Washington, 1999.

[2] R. Jain, R. Kasturi, B. G. Schinck *Machine Vision*. New York: McGraw-Hill, 1995.

[3] H. F. Liau, K. P. Seng, Y. W. Wong, L.-M. Ang *New Parallel Models for Face Recognition, pp. 306-309*. International Conference on Computational Intelligence and Security, Washington: IEEE Computer Society, 2003.

[4] P. J. Phillips, H. Moon, S. A. Rizvi, P. J. Rauss *The FERET Evaluation Methodology for face-recognition algorithms, pp. 1090-1103*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22(10), 2000.

[5] Y. Zana, R. M. Cesar *Face recognition based on polar frequency features, pp. 62-82*. ACM Transactions on Applied Perception, vol. 3(1), 2006.

[6] M. Lee, C. H. Park *An efficient image normalization method for face recognition under varying illuminations, pp. 128-133*. Proceeding of the 1st ACM international conference on Multimedia information retrieval, New York: ACM Press, 2008.

[7] L. Wiskott, J.-M. Fellous, N. Krueuger, C. von der Malsburg *Face Recognition by Elastic Bunch Graph Matching, pp. 775-779*. Transactions on Pattern Analysis and Machine Intelligence, vol. 19(7), 1997.

[8] W. Zhao, R. Chellappa, A. Rosenfeld, P.J. Phillips *Face Recognition: A Literature Survey, pp. 399-458*. ACM Computing Surveys, vol. 35(4), 2003.

[9] M. Turk, A. Pentland *Eigenfaces for Recognition, pp. 71-86*. of Cognitive Neurosicence, vol. 3(1), 1991.

[10] A. Pentland, B. Moghaddam, T. Starner *View-Based and Modular Eigenspaces for Face Recognition, pp. 84-91*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Seattle: IEEE Computer Society, 1994.

[11] H. Moon, P.J. Phillips *Computational and Performance aspects of PCA-based Face Recognition Algorithms, pp. 303-321*. Perception, vol. 30, 2001.

[12] A. Ferraz, E. Esposito, R.E. Bruns and N. Durn *The use of principal component analysis (PCA) for pattern recognition in Eucalyptus grandis wood biodegradation experiments, pp. 487-490*. Journal of Microbiology and Biotechnology, vol 14(4), 1998.

[13] R. Saegusa and S. Hashimoto *Pattern Recognition Using a Nonlinear PCA, pp. 73-80*. Proceedings of GVIP 05 Conference, Cairo: CICC, 2005.

[14] W. Zhao, A. Krishnaswamy, R. Chellappa, D.L. Swets, J. Weng *Discriminant Analysis of Principal Components for Face Recognition, pp. 73-85*. Face Recognition: From Theory to Applications, H. Wechsler, P.J. Phillips, V. Bruce, F.F. Soulie, T.S. Huang (eds.), Berlin: Springer-Verlag, 1998.

[15] A. M. Martinez, A.C. Kak *PCA versus LDA, pp. 228-233*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23(2), 2001.

[16] R.A. Fisher *The Statistical Utilization of Multiple Measurements, pp. 376-386*. Annals of Eugenics, vol. 8, 1938.

[17] C. Liu *Gabor-Based Kernel PCA with Fractional Power Polynomial Models for Face Recognition, pp. 572-581*. IEEE Transactions on Pattern Analysis and Machine Intelligenc, vol. 26(5), 2004.

[18] W. Li, W. Cheng *Face Recognition Based on Adaptively Weighted Gabor-LDA, pp. 130-134*. Proceedings of the 2008 Fourth International Conference on Natural Computation, Washington: IEEE Computer Society, vol. 4, 2008.

Usage of the webcam as 3D input device

Pavel Vlašánek
Supervised by: Alexej Kolcun

Faculty of Science
University of Ostrava
Ostrava / Czech Republic

Abstract

The paper is focused on using the webcam as a 3D input device. Generally it means the user may use any object and shadow of this object to obtain the coordinates x, y and z (position in the space). This idea allows to use a navigation in three dimensions without an expensive or atypical hardware. The system needs one ordinary webcam, an ordinary computer and a source of suitable light.

The goal of this paper is to describe an implementation of a software usable for handling an input from webcam with conditions described previously.

Keywords: navigation, shadow detection, foreground separation, webcam

1 Introduction

We know many devices that can be used for communication with a computer. A keyboard and a mouse are common but nowadays they may not be sufficient and comfortable. The reason is simple, each of these devices is a piece of hardware not allowing modifications. You can neither add the additional buttons into the mouse nor the new keys into the keyboard. The modern way for communication with a computer is the input from webcam. The user does not need a special controller very often and that is a big advantage. Good examples are Microsoft project Natal [1] and project Cam Space [2]. In the case that we do not need to use a depth we can use a system for a hand gestures recognition [3].

Instead of webcam and computer vision the special hardware for 3D navigation may be used. The way to control the computer by using a data glove and gestures [4] or multimodal input [5] is very natural. We can also use a special hardware for virtual finger tap. The company NOVIA AG [6] offers touchscreens with this functionality.

Basically, role of commonly used devices is allowing to work with the computer in the most natural way. The mouse is a good example because of using it as a hand. The operating system obtains position of the mouse and state of the buttons. We can say the mouse is 2D input device but via click we can provide something like a touch. The touch may be interpreted as a level of depth.

The main idea of this paper is shown in Figure 1. A finger works as a mouse (has a position) but we can also obtain better information about depth instead of a simple click.

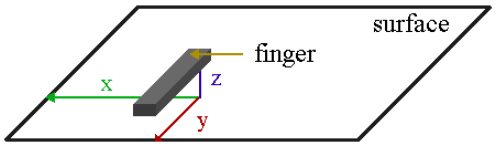


Figure 1: Image of the real scene

The parameters of a finger (or any another object) are a position (x and y) and the distance from the surface. The distance will be proxy for z coordinate allowing to work with the depth.

2 Previous works

The system needs to capture an image, detects an object and the shadow and then computes the coordinates dependently on mutual positions of those. I have studied lots of materials regarding object/shadow detection and chose a version via color (instead of e.g. contours). The first idea was to use HSV colour space because this space is more suitable for shadow detection [7]. At least RGB colour space was selected because we do not need to have a whole compact image of the object/shadow. The second reason was that surface colour(s) should be a very different comparing to object/shadow. The third reason was faster computing.

Detection based on the contours was explored too. I have experimented with Canny edge detector [8] but results were indicating another way of thinking. The solution based on contour detection instead of colour detection will be deeply investigated lately and is it a part of the future work.

The next problem was an foreground image. Colour values were very unstable and very often there was an edge between the object and the shadow practically invisible. The reason was that thresholds levels were very hardly setup. The pixels from object had a same value as the pix-

els of the shadow. This situation may causes some trouble, because shadow/object image may be very unclear. The first idea for improvement was to use an erode and dilation algorithms [9]. The results were not good enough so I have tried another idea. The meanshift image segmentation seems to be a better solution and it will be presented in chapter 4.2.

3 Hardware setup

There is the input image in Figure 2. The system needs an object, a surface, a source of the light and a webcam. From algorithmic point of view the system needs a mechanism for the foreground separation with some kind of smoothing, and for the shadow and the object recognition.

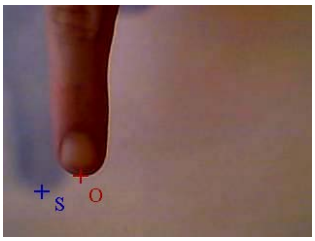


Figure 2: o point is a the end of finger, s point of the shadow

While the finger is falling down/raising up s point is coming closer/further to the o point. From these facts we can assume formula

$$h_l = |O_x - S_x|$$

where O_x is the x position of the finger and S_x is the x position of the thrown shadow. The formula above is valid (and we can assume that h_l is proportional to the z coordinate) for the scene with one shadow (or the strongest one among more shadows) coming closer/further to the object from the side. Practically the horizontal distance between points must be dependent on the real distance between object and surface. The relationship between the result h_l and z coordinate depends on the lighting. The comparison is shown in Figure 3.

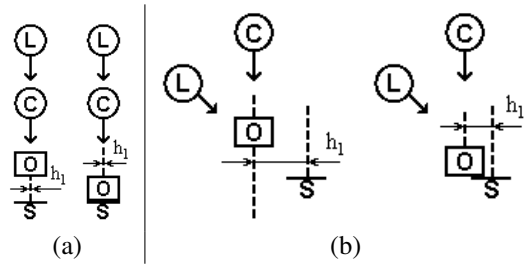


Figure 3: light L, camera C, object O and shadow S

If the shadow lies exactly under the object distance h_l is not affected by the distance between the object and surface. In Figure 3 (a) the distance h_l is roughly constant but in the second image (b) it depends on the height of the object over the surface. You can see that important thing is the mutual position of the camera and the light. There is shown a whole scene in Figure 4. Our goal is to express dependency between the height h and the distance between the object and the shadow as

$$\xi = \xi_1 + \xi_2$$

We can assume from triangle similarity that

$$\xi_1 = \frac{hx}{H-h}$$

$$\xi_2 = \frac{h(\Delta - x)}{H-h}$$

From these formulas we can assume the final formula

$$h = \frac{H\xi}{\Delta + \xi}$$

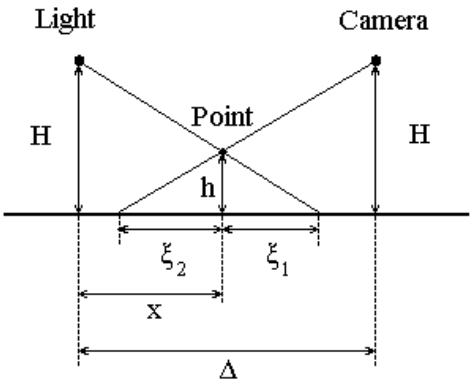


Figure 4: The exact view of the scene from up

There is dependency graph between h and Δ in Figure 5. There is shown an effects H to Δ. The outermost curve is for $\Delta = H/4$ and the lowest for $\Delta = 2H$. The increased Δ approaching a linear relationship thus linear formula above may be used instead of linear fractional. The linear formula is much simpler and faster and for more correct behavior variable h_l should be used as

$$h = h_l c$$

where c is the calibration constant[1].

4 Image processing

We briefly present the techniques used in the paper in this section. The output of these techniques will be more suitable image for processing than the original input from the webcam.

The system works as it is shown in the pseudo code below:

[1] A value depends on difference between the real height h and the measured height h_l.

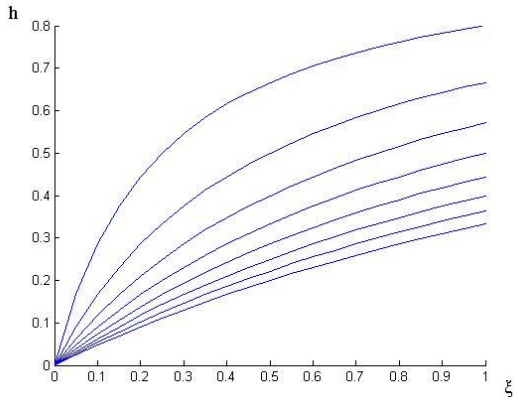


Figure 5: Dependency graph

```
determine_background();

while (!end())
{
    determine_foreground();
    smooth_foreground();
    create_images_IoIs();
    determine_the_points_OS();
    compute_coordinates();
    show_results();
}
```

The while cycle is repeated till the user stops the program.

4.1 The foreground separation

We are using an averaging background method which is suitable for static background and for scenes with the fairly constant lighting [10]. The average and averaged difference of each pixel are used in this method. As the first step the user has to provide a couple of images of the background. Then we can compute an average as

$$I_{avg} = \frac{\sum_{i=0}^{n} I_i}{n}$$

and the averaged difference as

$$I_{diff} = \frac{\sum_{i=1}^{n} |I_i - I_{i-1}|}{n}$$

where n is a number of the background images and I is the background image. Now we can create a model which will be used for the foreground detection. We will use the two thresholds defined as

$$I_{low} = I_{avg} - I_{diff} s_l$$

$$I_{high} = I_{avg} + I_{diff} s_h$$

where I_{low} is the lower threshold, I_{high} the higher one and s is for determine a range[2].

[2] $s_l = 6$, $s_h = 7$ according to [10].

After this analysis we can already process a new image from the webcam containing a foreground. Each of the pixels of I_{new} covered by

$$I_{low}[x,y] < I_{new}[x,y] < I_{high}[x,y]$$

is recognized as a foreground.

4.2 The meanshift image segmentation

The technique was used to simplify a foreground image. The pixels of the shadow have very similar values and we may suppose the same for the pixels of the object[3]. Situation may be different in the captured image because of the noise. Therefore the usage of the meanshift image segmentation is very suitable because it smooths colour variations [11].

The algorithm is based on meanshift clustering over the colour [10]. A meanshift window passes through the space and finds the groups with the highest density of data. Each point which is converging at a peak becomes connected by the peak. This ownership forms the segmentation of the image [10]. The Figure 6 shows a comparison of an intensity of the row of the image without this segmentation and with the segmentation.

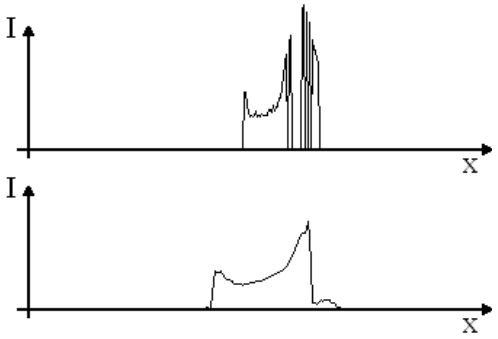


Figure 6: The intensity of the same line without meanshift segmentation (up) and with the segmentation (down)

Clusters with similar colour are replaced by the one colour. The shadow and the object are more recognizable and some noise is also eliminated.

4.3 The object detection and the shadow detection

Colours of the object (finger) and of the shadow have to be different than the background colour. The prerequisite is very important because of RGB colour space[4].

The finger has a value of the red channel higher than value of the green and the blue channel. All three values of the shadow are very similar. It seems that the blue channel

[3] Object with small colour variations is highly recommended.

[4] Segmentation is based on colours so shadow and object must be visible.

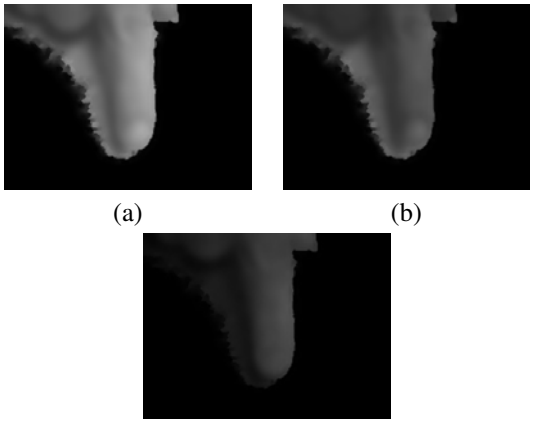
is not important so only red and green channels will be used.

$$I = \{I_r, I_g, I_b\}$$

The new image I_p is defined as follows

$$I_p = I_r - I_g$$

where I_r and I_g are channels from the image I mentioned above. The images related to this formula are in Figure 7[5] and you can see that each pixel in the shadow has a very similar colour and each pixel in the finger too.



(a) (b)

(c)

Figure 7: (a) I_r (the red channel of the foreground); (b) I_g (the green channel of the foreground); (c) I_p (difference between red and green channel)

In the chapter 4.4 there is explained how we can obtain an images of the object I_o and of the shadow I_s. The image I_s contains the white pixels represent a shadow and black pixels represent the rest. Similarly the white pixels in an image I_o represent an object and black pixels represent the rest. The points o and s from Figure 2 are the first white points in images I_s and I_o from right bottom.

In comparison with [3] our system does not need to know the contours. The main points are obtained from similar pixels recognized as shadow and from pixels recognized as object.

The user have to observed the rule about the object position. The system is depend on the pivot points thus the object have to be positioned as is shown in Figure 2. It means that the top of the object must come closer/further to the top part of the captured area.

4.4 Calibration

A suitable lighting source is very important prerequisite for the correct behaviour of the system. In other words there have to be a clearly visible shadow and an object (finger) in the image I_p as it is shown in Figure 7. In other words, after $I_r - I_g$ shadow and object must be visible. From observations we can say that floodlight is more appropriate than a sunlight.

The requirements for the calibration are very simple. We have to set thresholds of the finger and of the shadow. It will be used for separation from the foreground. Only four variables below have to be set:

- shadow low threshold T_{sl}
- shadow high threshold T_{sh}
- object low threshold T_{ol}
- object high threshold T_{oh}

The low/high thresholds mean the highest/lowest values of the pixels in the shadow or the object part. The range between the T_{sl} and the T_{sh} must be different compared to the range between the T_{ol} and the T_{oh}. We can obtain the images of the object I_o and of the shadow I_s as follows

$$I_o[x,y] = \begin{cases} 255 & T_{lo} < I_p[x,y] < T_{ho} \\ 0 & otherwise \end{cases}$$

$$I_s[x,y] = \begin{cases} 255 & T_{ls} < I_p[x,y] < T_{hs} \\ 0 & otherwise \end{cases}$$

In Figure 8 there is shown one row in I_p image. You can see the edge between the shadow and the object as rapid change in intensity. The pixels with lower intensities will be recognized as shadow where exact bounds are chosen via T_{sl} and T_{sh}. Similarly, the object will be represented by higher intensities defined by T_{ol} and T_{oh} variables.

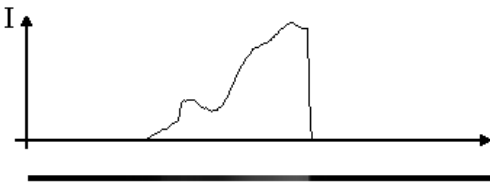


Figure 8: The intensity of the line of the I_p image.

5 Application

We are presenting the possible usage in this section. The application is written in C++ for graphic routines and obtaining images from a webcam an OpenCV library [12] was used. The user needs a computer, one ordinary webcam, a paper and a pencil.

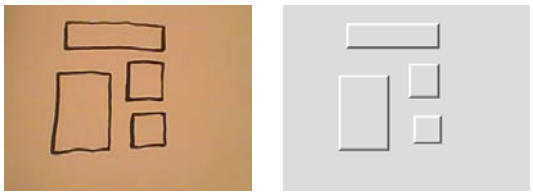
We have chosen a virtual keyboard for demonstration of the whole idea. The application provides handling of the key drawn by hand. The user may tap on the paper and virtually press the key and use it for comunication with the computer. The user may draw any number of the keys exactly how many really needs. For example the user will

[5] These images were captured with indoor lighting.

need a controller for any car game thus draw only key for acceleration, key for break and keys for turning. This solution has a big advantage because the user may choose count and position. The user may draw also a special controller as piano keyboard and on the touch react by playing a sound.

5.1 The keyboard detection

The user may draw any number of the keys as the arbitrary quadrangles. There are shown sketches and recognized keys in Figure 9.



(a) (b)

Figure 9: (a) Image of the four hand drawn keys; (b) Keys have been found and drawn as the rectangles

The recognition is based on the corners. We need only the strongest corners because there may occur a roughness on the line of the hand drawn key. At first we need to calculate minimal eigenvalue of gradient matrices using the Sobel operators (for more information [13], [14]). The results are stored to the I_{src}. Then non-maxima suppression is computed as follows

$$I_{dst}[x,y] = \begin{cases} I_{src}[x,y] & I_{src}[x,y] > mq \\ 0 & I_{src}[x,y] \leq mq \end{cases}$$

where I_{src} is a source image, I_{dst} is a destination image, m is the max value of the intensity obtained from I_{src} and q is quality level[6].

We have the corners as a result of the method above. The recognition starts with the left up point. Then we find the nearest point in the x-direction and the y-direction. The fourth point is the nearest point in the x-direction from the third point.

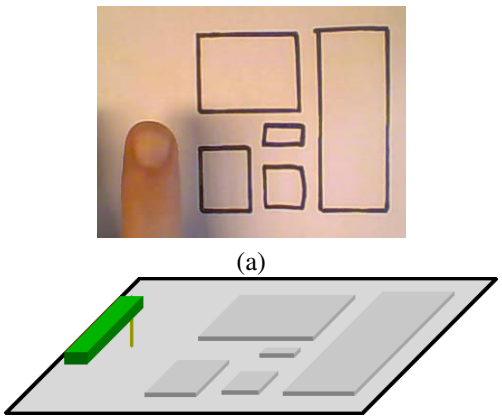
5.2 Graphic output

There was added an isometric view for better navigation in the 3D space. Figure 10 shows 2D input image captured from the webcam. It is drawn as an isometric image with some level of depth.

6 Results

We are able to provide the navigation in the 3D space using one webcam. The navigation may be controlled via

[6] $q = 0.2$ according to [10].



(a) (b)

Figure 10: (a) The original image; (b) Isometric view drawn

object and the shadow of this object. The user has to show a background using a source of the light for obtaining a clearly visible image of the foreground composed from the object and the shadow. The places with very similar gray colours (according to Figure 7 (c)) are recognized as the object and the other one as the shadow using the thresholds. Middle top points of these places are chosen as the pivot points. The distance between the points is used as the level of depth and the position of the object point as x and y coordinates in Cartesian xyz-system. The coordinate z may be computed via linear or linear fractional depends on the position of the light and webcam. The computing of these coordinates works fast enough and it is usable for the usage in the real time.

The system has been tested in home conditions. It means that distances H and Δ from Figure 4 has been in tens of centimeters. Additionally the distance Δ was much longer than H thus linear formula was used. If the user set the suitable c then value of the h corresponds with real distance between the object and the surface.

7 Conclusions and future work

Functionality of the system was shown on virtual keyboard example where the level of the depth (z coordinate) is used for click detection. The virtual click is only one aspect of the possible usage and it has been chosen because of not very demanding dependencies on lightness and calibration. We can expand the idea of the virtual click using the projector showing an image on the wall. According to Figure 4 the projector is the *Light*. From the foreground image obtained as described above a hand and the main finger are separated. The user may tap on the wall and press the button shown by the projector. Then a computer obtains a press event and handles it by a particular way.

As mentioned above a virtual click is only one possible usage. In general the system may be used as a navigation in the 3D space using an object as the controller. The con-

troller may be used for 3D modeling and via gesture we can create and grab the virtual points and than make the models, move models, and so on.

We can also use a simple object as a virtual pencil and draw via touching the wall. The webcam obtains an image. On the places where the pencil is touching the wall the program draws the points. This idea may be expand by third dimension. The distance between object and the surface will be used for computing of the z coordinate as is shown in chapter 1. The user have to set an specific event to begin drawing and event for stop. We have the webcam and the system for the computer vision thus usage of gesture is very suitable [3]. For example the user may join index finger and thumb and than via the object (or the finger) draw in space. The system will be interpreted as path of the object as colour track and then when the user makes a stop gesture drawing will be stopped.

The work on this project is still ongoing and these ideas presented in this chapter will be the main targets in the future.

References

[1] "Project natal." `http://www.xbox.com/en-us/live/projectnatal/`. visited: 28.3.2010.

[2] "Camspace." `http://www.camspace.com/`. visited: 28.3.2010.

[3] J. Cíger and J. Plaček, "The hand as an ultimate tool," *SCCG*, 2000.

[4] E. Sánchez-Nielsen, L. Antón-Canalís, and M. Hernández-Tejera, "Hand gesture recognition for human-machine interaction," *WSCG*, vol. 12, 2004.

[5] J. García, J. P. Molina, D. Martínez, A. S. García, P. González, and J. Vanderdonckt, "Prototyping and evaluating glove-based multimodal interfaces," *Journal on Multimodal User Interfaces*, vol. 2, 2008.

[6] "Interactive devices." `http://www.novia.ch`. visited: 28.3.2010.

[7] "Shadow detection." `http://dali.mty.itesm.mx/~autonomos/Navdyn/node11.html`. visited: 28.3.2010.

[8] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, 1986.

[9] J. R. Parker, *Algorithms for image processing and computer vision*. John Wiley & Sons, 1997.

[10] G. Bradski and A. Kaehler, *Learning OpenCV*. Gravenstein Highway North: O'Reilly, 2008.

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

[12] "Open source computer vision." `http://opencv.willowgarage.com/wiki/`. visited: 28.3.2010.

[13] M. Nixon and A. Aguado, *Feature Extraction & Image Processing*. Linacre House: Elsevier, 2008.

[14] "Opencv feature detection." `http://opencv.willowgarage.com/documentation/cpp/feature_detection.html`. visited: 28.3.2010.

Computer-Vision based Pharmaceutical Pill Recognition on Mobile Phones

Andreas Hartl*
Supervised by: Clemens Arth†

Institute for Computer Graphics and Vision
Graz University of Technology
Graz / Austria

Abstract

In this work we present a mobile computer vision system which simplifies the task of identifying pharmaceutical pills. A single input image of pills on a special marker-based target is processed by an efficient method for object segmentation on structured background. Estimators for the object properties size, shape and color deliver parameters that can be used for querying an online database about an unknown pill. A prototype application is constructed using the Studierstube ES framework, which allows to perform pill recognition on off-the-shelf mobile phones. System runtime and retrieval performance with the estimated features is subsequently evaluated on a realistic test set. The retrieval performance on the exemplarily used Identa database confirms that the system can facilitate the task of mobile pill recognition in a realistic scenario.

Keywords: pill recognition, mobile phones, marker, feature estimation, database

1 Introduction

Correct and timely identification of drugs which are encountered without packaging, is a major problem in healthcare. If the exact type of drug taken by a patient is known, a more directed therapy can be applied and unnecessary medical effort can be avoided. The identification of pharmaceutical pills like tablets, capsules and pills (dragées) must often be performed as fast as possible. If pills need to be identified on the way, only a visual inspection is possible. This task can be supported by performing computer vision on mobile devices. The ubiquitous nature of mobile phones, the built-in camera as well as advancements in processing power make up an attractive platform for application development. The main challenges can be seen in the changing environment of operation as well as the limited computational resources of mobile devices.
Some domain research reveals three main methods for pill identification: (1) look-up in a book, (2) application of a special identification scheme, (3) querying a database. From the aforementioned methods, an identification scheme offers best accuracy within a reasonable amount of time. In mobile applications, the availability of an identification method and the necessary skill of the operator are critical. This drawback may be mitigated by using mobile phones for pill identification.
In a visual identification system a set of features must be determined. These could be shape, size, color, scores and imprints, but also transparency or texture. Such features can be estimated directly on the device or on a server which is used to process an image taken by the mobile phone. It is important to note that mobile network coverage is still imperfect, which can render a solution that employs server-side processing entirely useless. In the proposed approach, properties of pills may be instantly estimated without requiring a network connection. Thus, features such as object size can still be used to identify pills by other means. Besides, on-device feature estimation is an essential building block for a truly independent solution in which pill information is stored directly on the device.
In the following we present a system for instant visual pill recognition on mobile phones based on an image taken with the built-in camera. Initially, a set of features is estimated for each individual pill. This information is used to query an online available database. Then, a series of candidates is presented on the mobile phone along with additional information. In this work the features shape, size and color are estimated since they are supported by freely available online databases.
The remainder of this work is structured in the following way: In Section 2 a short review of related work in the area of pill recognition as well as computer vision on mobile phones is presented. Section 3 is devoted to segmentation, as it is a fundamental problem in the application to be developed. Section 4 deals with the feature extraction process from the segmented regions. In Section 5 an overview of a prototype implementation for mobile devices is given, followed by an evaluation of the system on an exemplary database in Section 6. A conclusion and additional remarks are given in Section 7.

*ahartl@student.tugraz.at
†arth@icg.tugraz.at

2 Related Work

To the best of our knowledge there is only a single report of a pill recognition system so far, which is described in a US patent [13]. It deals with an automatic method for the task of verification that the content of a container filled by a dispensing system, corresponds to a given prescription. Little knowledge is available about the inner workings of the system. According to the given source, this is a fixed system which uses a frame grabber as input and employs color, geometry and surface features to identify all pills that a given machine may process. Prior knowledge is used to simplify the problem which gives better results.
Ubiquitous mobile devices are used for various tasks in computer vision. Wagner et al. perform robust 6DOF natural feature tracking using modified SIFT and Ferns as descriptors [11]. In order to allow computation on mobile devices, extensive modifications are carried out to the basic concepts of SIFT and Ferns, followed by an instructive evaluation of system performance. Despite severe limitations in processing speed and memory bandwidth, they achieve real-time performance when using textured planar targets on current-generation phones.
Klein and Murray present a system for parallel tracking and mapping on camera phones [7]. They implement a key-frame based SLAM system that is capable of generating and augmenting small maps. Limited computational resources and problems in image acquisition such as a rolling shutter are specifically accounted for to allow computation on mobile devices (iPhone 3G).
In the system at hand robust estimation of object features from a single input image must be performed. Due to the nature of the problem, it seems justified to run through a separated segmentation, feature extraction and classification step (depending on the feature), in which tasks may be optimized independently. For each of the steps needed in our application a vast amount of literature is available, but self-contained work on how to perform these steps efficiently on mobile devices is not available. Thus, we propose a series of solutions which make the problem computable on current mobile phones in instant time.

3 Robust Segmentation

Segmentation is a critical problem in the system at hand, because the results determine the quality of any subsequent feature estimation task. As the setup used for image acquisition is not fixed and lighting conditions may vary, performance with changes in scale, perspective distortion and non-uniform lighting is particularly important. Since it is necessary to estimate the size of arbitrarily colored pills, a marker-based target of known geometry and background is used. For reasons of practicability, the dimensions of this target are chosen to fit into a wallet. Using a checkerboard background (see Figure 1), it is possible to robustly segment objects with reasonable requirements in processing speed. Applicable images may be obtained from an autofocus camera, since fixed focus cameras are unable to give a sharp image of suitable size.

3.1 Reduction of Input Data

The marker-based target can also be used to reduce the amount of input data for segmentation. Besides, feedback about the image geometry is possible before further computations take place. This may be helpful, because rectification of the entire image is not efficient. For image regions, rectification is feasible, however, because the amount of input data is much smaller.
In the problem at hand, a homography between points on the image plane and points lying on a plane defined by the dimensions of the target can be computed. This bijective mapping of coordinates between planes can be represented by a non-singular matrix of size 3x3 with 8 degrees of freedom (see e.g. the work of Hartley and Zisserman [6]). For the computation (estimation) of the 3x3 Matrix $\mathbf{H_{IW}}$ ($\mathbf{H_{WI}} = \mathbf{H_{IW}}^{-1}$) at least $n = 4$ point correspondences must be found between image plane and the corresponding world plane. In consideration of a possibility to realize such a rectification procedure efficiently, the minimum number of 4 points is chosen. They correspond to the corners of the reference rectangle of the marker-based target. Thus, a rectangular region of interest (ROI) can be defined in the image plane using the known location and dimensions of the checkerboard area and the previously computed homography.

Figure 1: Frame marker with structured background

3.2 Segmentation Algorithm

A segmentation of objects on the proposed card can be obtained by local adaptive thresholding and morphological operations on a grayscale input image. All relevant computations may be carried out on a square neighborhood that extends into both directions w_h pixels, giving a minimum total square length of $2 \cdot w_h + 1$ pixels. The principal

algorithm is as follows:

$$M_{seg} = (\neg(M_{Th_1} \bullet SE_1) \circ SE_2) \bullet SE_1, \qquad (1)$$

where M_{Th_1} denotes a mask obtained by local adaptive thresholding with a neighborhood size of $2 \cdot w_h + 1$, and SE_1 and SE_2 denote structuring elements of length $2 \cdot w_h + 1$ and $2 \cdot w_h + 3$. The symbol $\neg$ is used for mathematical inversion and the symbols $\circ$ and $\bullet$ denote morphological opening and closing. In our implementation, the checkerboard pattern is detected by application of the proposed procedure using an efficient method for local adaptive thresholding by Shafait et al [9].
Further processing is necessary to extract smooth contours. Thus, region labeling with integrated boundary computation is carried out after the work of Chang et al. [3], in which a linear-time method is described. Subsequently, the convex hull of each boundary is computed using the algorithm by Graham [5]. In the final step, we apply variant of the line drawing algorithm by Bresenham [2] to obtain connected chains (see Figure 2). If necessary, the region may be calculated in a flood fill operation then. It must be noted that the choice of the single parameter w_h is uncritical in the application at hand. However, the length of the checkerboard pattern must be chosen with care, since it determines the quality of segmentation to a large extent. This choice is dependent on the desired resolution, since a sufficiently sharp image of this pattern is required. Currently the length is set to $0.6mm$, favoring images of 640 x 480 pixels, as a compromise between accuracy, runtime and usability.

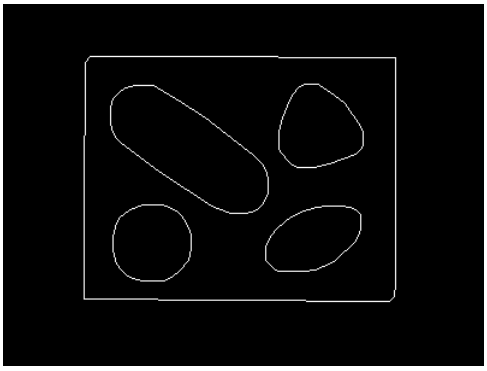


Figure 2: Segmentation borders for Figure 1 (extract)

4 Feature Estimation

In the application at hand we measure objects and perform a coarse estimation of shape and color. The following considerations are based on the assumption that the starting position for property estimation is the boundary of an image region that resembles the silhouette of an object as closely as possible. For each step, an efficient solution is needed to allow for computation on mobile devices.

4.1 Size Estimation

Geometric properties of pharmaceutical pills such as length, width and height are important features for identification (see Section 6). Invariance to perspective distortion is necessary to allow for correct measurements. With a single input image, measurements can be carried out in two dimensions by using an available homography (see Section 3.1). In the current approach for size estimation, distortions in image acquisition are not considered, since the error caused by segmentation is assumed to be dominant.

4.1.1 Measurement of Length and Width

In the following, the length of a pill is defined as the extension of its boundary along the major axis of its perpendicular projection. This suggests that the extension in the perpendicular direction is defined as width. With this definition the majority of pill shapes may be measured correctly without further interaction. Consequently it seems reasonable to determine the direction of maximum variance within a region where a pill is supposed to be. To achieve this goal, we propose a procedure in which a subset of all region points is used for increased robustness. First, a flood fill operation is carried out on the contour and a suitable number of points is selected and rectified using an available homography. Subsequently, the direction of maximum variance $\mathbf{v_1} = [v_1; v_2]$ is determined by analysis of the covariance matrix $\mathbf{C}$ [12]. In order to obtain a more accurate estimate for the length and width of an object, it is necessary to project the boundary that makes up an object using the obtained vector $\mathbf{v_1}$. In the absence of outliers, it is sufficient to project the world points of the convex hull, because they bound the extension of the object. If the vector $\mathbf{m} = [m_x; m_y]$ corresponds to the centroid of all world plane points that make up a region, a projection may be computed as:

$$\begin{bmatrix} xi_{W(p)} \\ yi_{W(p)} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \\ -v_2 & v_1 \end{bmatrix} \begin{bmatrix} xi_W - m_x \\ yi_W - m_y \end{bmatrix}, \qquad (2)$$

where $[xi_W; yi_W]$ denotes a point in the world plane that belongs to the current region, and $[xi_{W(p)}; yi_{W(p)}]$ denotes its projected counterpart. An estimation of length and width may then be obtained by computation of differences between minimum and maximum values of these coordinates (see Figure 3 for an illustration of this process).

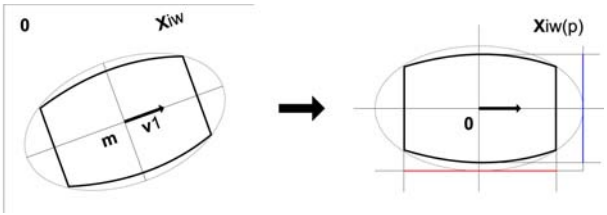


Figure 3: Projecting boundary points (world plane)

4.1.2 Considerations on Accuracy

The square size sq_w of the checkerboard pattern, that limits the accuracy of segmentation, is identified as the major source of error. Under the assumption of a target that entirely fills the viewable area and does not exhibit other kinds of distortion, the expected error is dependent on the square size sq_w. As an upper bound, the error may be assumed to be double the square size then. The minimum usable square size sq_w is dependent on the image resolution. If we assume that pills may be correctly segmented under the previous conditions, the maximum error e_{max} can be estimated as follows:

$$e_{max} = 2 \cdot sq_w \tag{3}$$

With a square size of $0.6mm$, the expected error e_{max} is $1.2mm$. With images of 640 x 480 pixels, the square size may be reduced to $0.5mm$, giving an estimated error e_{max} of about $1mm$. Using an ordinary ruler as measurement tool, it is rather difficult to measure lengths below 1 mm, and the shapes of pills further aggravate measurements. For these reasons, the accuracy of the proposed method can be expected to be better than what can be obtained with a ruler.

4.2 Shape Estimation

For reasons of efficiency, the boundary of an object is chosen to serve as a basis for the estimation of object shape. A shape estimator should be as invariant to changes in translation, rotation and scale as possible. A certain degree of perspective distortion should still be tolerable to account for artifacts. In the following we use a modified pairwise geometric histogram (PGH) as shape descriptor and subsequently perform shape matching.

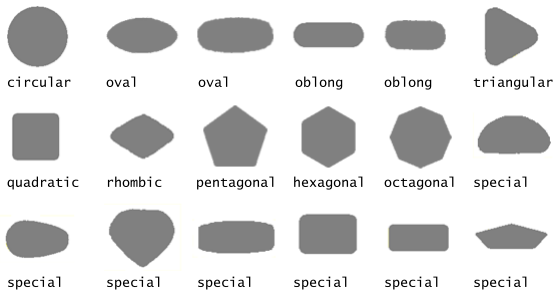


Figure 4: Examples of pill shapes with classes

An analysis of available samples as well as examples from literature led to the decision to categorize each example into one of 9 representative shape classes containing only **convex** instances. A class termed *special* is added to represent shapes that do not fall into any of the other categories (see Figure 4). Shapes of pharmaceutical pills show a pronounced intra-class variance as well as partially low inter-class variance. Thus, an applicable shape descriptor needs to have considerable discriminative power. We propose a modified pairwise geometric histogram (PGH) for this task, which is efficiently computable and sufficiently invariant to changes in scale. This development of a PGH also allows for efficient shape matching.

4.2.1 Modified Pairwise Geometric Histogram

In the PGH descriptor oriented line segments are investigated (see the work of Evans et al. [4]). Their relative orientation and perpendicular distance is analyzed and this information is collected in a 2D histogram of size $D \cdot A$. The set of possible angles and distances is mapped onto the histogram by accumulation of occurrences. During the process, each line is used as a reference line and the angle, as well as the perpendicular distance is computed to all the other lines. In fact, every line is represented as a histogram and the accumulation of all histograms represents the shape of the object. An exemplary histogram is shown in Figure 5. Although mainly designed for polygo-

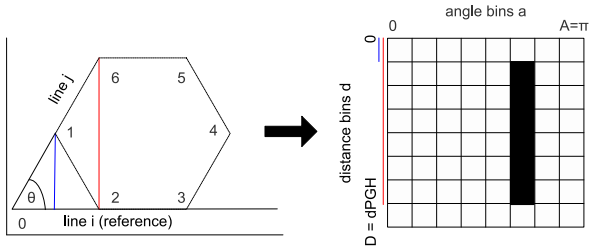


Figure 5: PGH computation of a single line

nal shapes, application to non-polygonal types is possible, if a polygonal approximation is computed as an initial step. Besides, the PGH is not scale invariant in its original form. Limited scale invariance may be achieved by application of a suitable and stable similarity metric[1]. For reasons of efficiency, we perform the following method to achieve invariance for a limited range of scales: We assume that the maximum distance for a PGH can be estimated from the rectified contour. As the computation of the convex hull is part of the segmentation algorithm, no outliers may be expected. So, a measure of scale can be found just by searching for the maximum distance d_{max} in the key from their centroid. The maximum PGH distance may be computed as

$$d_{PGH} = 2 \cdot d_{max} \tag{4}$$

then. As the vast majority of pharmaceutical pills is convex, the PGH is constructed with a reasonable amount of bins using just the points which make up the convex hull. This corresponds to considerable savings in computation.

4.2.2 Classification

Due to the achieved scale invariance, we perform shape matching on a set of predefined samples using the Euclidean distance as a similarity metric. Although the possibly large size of the PGH may still be prohibitive for certain applications, this drawback can be mitigated by application of a suitable data structure.

4.3 Color Estimation

A color estimation procedure should correspond to human perception as good as possible. Besides, the procedure should be able to give reasonable results with varying lighting conditions. In general, pharmaceutical pills may have arbitrary colors (transparent pills are not considered). Research in literature and online databases reveals that most pills have one or two colors, out of a set of 16 basic color tones (black, white and gray are treated like a color). These are black, white, blue, beige, brown, gray, green, ocher, pink, violet, orange, peach, rose, red, cyan and yellow (see Figure 6 for a set of mean colors).

Figure 6: sRGB mean colors for pill color classes

For color estimation, the influence of lighting is reduced by applying a method for local white balance which is based on reference measurements on the marker-based target and subsequent scaling in input space. Due to the small amount of available samples and for reasons of efficiency, color estimation is based on a sRGB lookup table (LUT) which is created by evaluation of the ΔE_{CIE00} color distance metric (see e.g. the work of Vik [10]) in CIE LAB space. Per pixel classification results for a pill can be aggregated in a histogram. An analysis of this information gives the color estimation result.

4.3.1 Look-up Table Computation

Lookup tables allow fast classification at the cost of additional storage and memory requirements. For practical reasons, the LUT size should be comparatively small. In the following, the value s_{LUT} denotes the amount of entries for each channel in a 3D LUT. We propose the following method for the creation of a sRGB LUT that resembles human perception of color, from a small number of samples per class:

- Definition of representative sample colors c_{ij} per class. They need to be visually similar and should show a smooth course of color within a class.
- Assignment of a label l_i to the examples of class i.
- Definition of the desired LUT size s_{LUT} (equal for all dimensions).
- Conversion of all sRGB sample colors into CIE LAB space using the D65 white point (daylight).
- Iteration through all sRGB LUT entries, computation of the corresponding sRGB color in CIE LAB space and the distances d_{nn1} and d_{nn2} to the nearest samples by using the ΔE_{CIE00} color distance metric.
- Assignment of a label l to the current LUT entry after analysis of the distances d_{nn1} and d_{nn2}.

Cases, where a decision is not possible or no suitable color may be found, are handled by suitable thresholds. The proposed approach relies on the assumption that the colors which may be traversed by exhibiting the step size that is defined by s_{LUT} on each channel, show no or negligible perceptual difference.

4.3.2 Color Classification

A classification for up to two colors may be obtained by analysis of the class histogram h_c of per pixel results on white balanced data. Thus, the relative amount of covered pixels in the region as well as the significance of the result, which is based on the distance between the first two entries of a sorted class histogram h_{cd} (descending order), is evaluated. The following metrics are used to decide on the color(s) of a region:

- coverage for a result with one color (label i):

$$c_i = \frac{h_{cd}(0)}{\sum h_c} \tag{5}$$

- coverage for a result with two colors (labels i and j):

$$c_{i,j} = \frac{(h_{cd}(0) + h_{cd}(1))}{\sum h_c} \tag{6}$$

- significance for a result with one color (label i):

$$s_i = \frac{h_{cd}(0) - h_{cd}(1)}{h_{cd}(0)} \tag{7}$$

- significance for a result with two colors (labels i,j):

$$s_{i,j} = \frac{1}{s_i} \tag{8}$$

A decision is possible, if one of these coverages is above a threshold c_{th}, because this assures a more meaningful result. Then, the products $c_i \cdot s_i$ and $c_{i,j} \cdot s_{i,j}$ are used to decide, whether one or more colors are present. If the first product is greater than the second, the result is the corresponding label i of the maximum entry in h_{cd}. Otherwise two colors are assumed and the labels i, j that correspond to the first two entries in h_{cd} are reported.

5 Mobile Phone Prototype

We implemented a prototype system suitable for mobile devices using the algorithms described in the previous chapters as well as existing functionality from the StbES framework [8]. It allows to obtain an estimate of recognition performance on mobile devices when querying an online available database. In Figure 7 an overview of the basic components is given. The application frontend may run on a Windows Mobile equipped camera phone as well as on a Windows PC. The database connection is decoupled from the application and may be adapted to arbitrary sources of information.

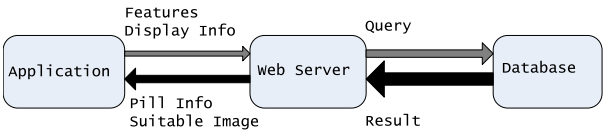


Figure 7: Basic system components

All computer vision related processing takes place on the mobile device (see Figure 8 for an illustration of the required steps). An exemplary pill recognition session is shown in Figure 9 A, B and C. The input image is obtained from live video, using the tracker of the given framework for target detection. The result of segmentation and color estimation as well as the obtained directions of measurement may be instantly verified from overlaid data (see Figure 9 A). In the pill browser, manual correction of query parameters is possible using just the directional stick of the mobile device (see Figure 9 B). Alternatively, input from a touchscreen is possible. In case of the evaluated online database an image for visual verification is provided, along with name, manufacturer, dimensions and mass of the candidate (see Figure 9 C). In order to reduce the initial round-trip time, only textual information is transferred at first and images are cached on the server for on demand retrieval.

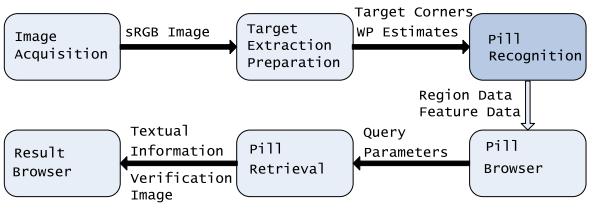


Figure 8: Course of the pill recognition application

Optimizations include excessive preallocation of memory, avoidance of floating point operations or use of fixed-point types. The retrieval of pill information is possible through any of the built-in communication facilities.

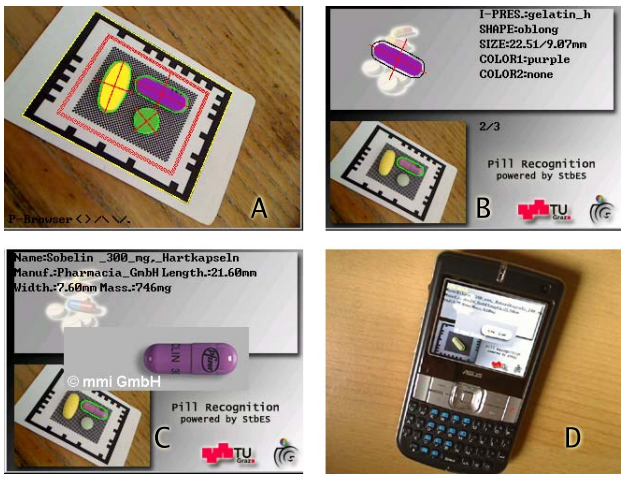


Figure 9: A feature estimation result, B pill browser, C result browser, D test with mobile phone

6 Evaluation

We evaluated the system in terms of estimator performance with suitable test sets that resemble typical operating conditions. Pill identification performance is evaluated with optimal estimator parameters using a set of manually classified examples as well as the Identa[1] online database. The latter is chosen to serve as the backbone for the application, because access is free. Thus, training of feature estimators needs to comply with Identa categories. Only a subset of possible pill shapes are supported as query parameters. The shape categories represent circular, oval, oblong and special shapes (see Figure 10). The created general shape data for pills was subsequently categorized to be in accordance with the shape classes of Identa. In the Identa query form, 14 colors are differentiated by their names as well as a small number of visually similar tones. For reasons of robustness, the tones *rose* and *peach* are merged. The samples for each class are taken from previously collected data on pill color. The Identa database is not fully consistent concerning shape and color, however. Mobile performance is evaluated on an Asus M530w smartphone[2], running Windows Mobile 6 (see Figure 9 D). It features a 2 mega pixel autofocus camera, a 416MHz fixed-point CPU, 64 MB of RAM and 256 MB Flash.

6.1 Datasets

In the proposed evaluation procedure, reference data for the captured pills is necessary. For this purpose, each sample was manually classified into the appropriate categories

[1] http://www.gelbe-liste.de/pharmindex/identa
[2] http://www.asus.com

Figure 10: Examples for Identa shape classes: A special class; B oval class; C oblong class; D circular class

for shape and color. The length and width of each sample were measured using a nonius. All collected information about the samples was subsequently stored for later use (see Table 1 for a description of contents). Despite its small size, the set contains pharmaceutical pills with the most current shapes, colors and dimensions. In Table 2 a detailed listing of the distribution of colors for single-colored pills is given. Test images for each example in

Shapes	*circular*	*oval*	*oblong*	*special*
Instances	41	26	33	8
Colors	*single*	*multi*		
Instances	98	10		
Sizes [mm]	*min. length*	*min width*	*max length*	*max width*
	5.68	5.68	18.07	18.07

Table 1: Contents of the reference set: shape, colors, size

the global reference set were captured at 640 x 480 pixels with differing lighting conditions. A labeled example is obtained by storing each filename at its corresponding entry in the global reference database. This means that one representative image for each sample in the global reference set (108 samples) is included for set D (daylight) and set F (fluorescent lighting).

Color	*black*	*white*	*blue*	*beige*	*brown*
Instances	1	29	6	11	4
Color	*gray*	*green*	*ocher*	*pink/violet*	*orange*
Instances	1	5	6	3	6
Color	*rose/peach*	*red*	*yellow*	-	-
Instances	11	5	10	-	-

Table 2: Contents of the reference set: single-colored pills

6.2 Results

Initially shape and color estimators were extensively evaluated to obtain optimal parameters (shape PGH: D=12, A=12; color LUT: $s_{LUT} = 36$). In this step, the best recognition rate for shape is 0.89 and that for color is 0.84, with little dependence on lighting. In size estimation the average deviation is 0.43*mm* (maximum deviation: 1.37*mm*). Runtime on the mobile device was evaluated for a pill of average size and for the largest available pill (see Table 3 for a detailed listing). For this purpose, the input image is read from a file. Segmentation consumes the largest portion of runtime, but the problem may still be computed on the mobile test device in a reasonable amount of time.

Task	**RT [ms] (largest)**	**RT [ms](average)**
overall	**1205**	**1114**
preparation	20	18
segmentation	698	714
shape features	204	213
shape class.	6	4
size estim.	44	29
color estim.	233	136

Table 3: Runtime evaluation on specific pills (640 x 480 pixels): results for largest pill in set DF as well as an average pill (optimum settings; rounded), RT...runtime

6.2.1 Pill Recognition

Due to the small amount of samples, pill recognition performance is evaluated in two different ways. First, the created global reference database is used as the source of pill information. In the definition of a query range for length and width, a deviation of 0.7mm is assumed. In the second part, pill retrieval performance is evaluated with Identa using a filtered set of 27 pills, that can be positively assigned to instances within Identa. In this case a correction of parameters by a human operator is assumed.

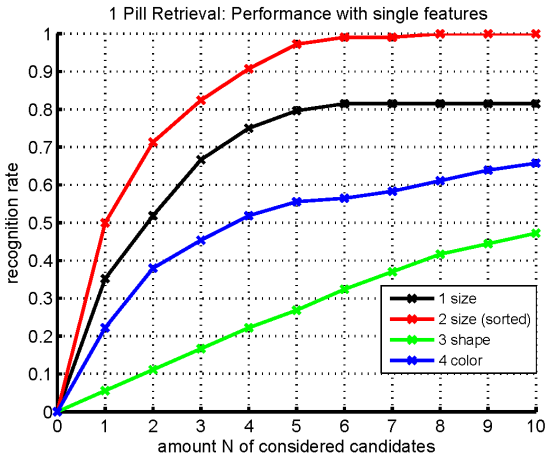


Figure 11: Pill retrieval on the global reference set (single features): results for *set D*

Retrieval performance for single features gives information about the descriptive power of a feature (see Figure 11). This means that *size* is the most powerful feature, followed by *color* and *shape* because higher recognition rates are achieved with a smaller amount of considered candidates. *Size* and *color* have a steeper slope and give

better results faster than the feature *shape*. Performance with the feature *size* reaches a recognition rate of 0.814 ($N = 6$ candidates), when querying using a range defined by the measurements and the expected accuracy in size. Performance of feature *size* improves considerably, if results are ordered by the minimum sum of deviations in length and width. Peak performance may drop when using several features.The reason may be mutual reactions of errors from the individual feature estimators, because their results were not corrected. We have included the results for a very conservative query strategy, using no prior information about the set and reporting the result of sequential querying for features. In Table 4 a summary of results is given.
When querying the Identa database using the filtered set, the final retrieval performance is 85.19 percent. On average, $N = 8$ candidates must be inspected to obtain the correct match (Identa contents 12/2009).

Size	Size (sorted)	Shape	Color	RR_{max}	#
x				0.8148	6
	x			**1.0000**	8
		x		0.4722	10
			x	0.6574	10
x		x		**0.7315**	6
x			x	0.6389	3
x		x	x	0.5556	3
	x	x		**0.8889**	5
	x		x	0.8148	2
	x	x	x	0.7037	2

Table 4: Pill retrieval on the global reference set (*set D*, $N = 10$): best results, RR...recognition rate

7 Conclusions

In this work we present a mobile pill recognition system for conventional smartphones. An algorithm for segmentation of pills from a business card sized marker is presented. Hereafter, relevant properties, such as size, shape and color are chosen to be determined by means of CV. A domain analysis is carried out to identify the most relevant shapes and colors of pills. The estimated features can be used for identification of medical pills within a database then. An example of this application interfacing an online-available database for medical matters is presented, followed by an extensive evaluation of the system in terms of algorithm speed and pill retrieval performance.
The obtained results show that the approach is able to work in instant time using commonly available smartphone hardware. Furthermore, the retrieval performance of the system on the exemplarily used Identa database confirms that the system can facilitate the task of visual pill recognition in a realistic scenario.
For the verification of practical use an extensive user study will be necessary. The incorporation of additional features such as brick lines or text could be the topic of future work. In the context of a human operator and higher runtime, the benefit must be evaluated. If the pill database is stored on the mobile device, a fully independent solution is possible.

References

[1] A.P. Ashbrook, N.A. Thacker, P.I. Rockett, and C.I. Brown. Robust Recognition of Scaled Shapes using Pairwise Geometric Histograms. In *Proceedings of BMVC*, pages 503–512, 1995.

[2] J. E. Bresenham. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal*, 4(1):25–30, 1 1965.

[3] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu. A Linear-Time Component-Labeling Algorithm using Contour Tracing Technique. *Computer Vision and Image Understanding*, 93(2):206–220, 2004.

[4] A.C. Evans, N.A. Thacker, and J.E.W. Mayhew. Pairwise Represenations of Shape. In *Proceedings of the 11th ICPR*, pages 133–136, 1992.

[5] R.L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972.

[6] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2008.

[7] G. Klein and D. Murray. Parallel Tracking and Mapping on a Camera Phone. In *Proceedings of ISMAR'09*, pages 83–86, 2009.

[8] D. Schmalstieg and D. Wagner. Experiences with Handheld Augmented Reality. In *Proceedings of ISMAR'07*, pages 1–13, 2007.

[9] F. Shafait, D. Keysers, and T. Breuel. Efficient Implementation of Local Adaptive Thresholding Techniques Using Integral Images. In *Proceedings of SPIE*, 2008.

[10] M. VIK. Industrial Colour Difference Evaluation: LCAM Textile Data. In *Proceedings of AIC'04*, pages 138–142, 2004.

[11] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose Tracking from Natural Features on Mobile Phones. In *Proceedings of ISMAR'08*, pages 125–134, 2008.

[12] S. Wijewickrema and A. P. Paplinski. Principal Component Analysis for the Approximation of a Fruit as an Ellipse, 2004.

[13] J. R. Wootton, V. V. Reznack, and G. Hobson. US Patent 6535637 - Pharmaceutical Pill Recognition and Verification system, 2003.

Segmentation and classification of fine art paintings

Zuzana Haladova *
Supervised by: Elena Sikudova†

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

Abstract

Since the development of the first text-based image search on the internet, the area of image retrieval has come a long way to sophisticated content based image retrieval systems. On the other hand, the semantic gap causes that it is still not possible to create a system which can correctly identify any object in the image. However, this paper proposes a solution for classifying the one sort of objects - paintings. This approach includes segmentation of the painting from the image, creation of the descriptor file from the segmented painting, and classification of the painting by matching its descriptor file to the created database of descriptor files of original paintings. The segmentation of the painting is achieved with 3 preprocessing steps, followed by adjusted Hough transformation. For the estimation of key points and creation of the descriptor file, the SIFT (Scalable Invariant Feature Transform) or the SURF(Speeded Up Robust Features) technique is used. The performance of both techniques is validated within the paper. The solution proposed in this paper was tested on the database of 100 Rembrandt Harmenszoon van Rijn's paintings.

Keywords: Fine art paintings, Segmentation, Classification, SIFT, SURF

1 Introduction

This paper interlopes three scientific areas, image retrieval and the classification and the digital preservation of art. The approach proposed in this paper consists of a CBIR (Content based image retrieval system) which operates over a database of fine art paintings. This CBIR system is defined (according to the categorization proposed by Data et al. [6]) as a system operating over domain specific collection, which queries by an image, with content-based processing of the query.

The motivation for the creation of the system are deficient retrieval possibilities in the most art web-galleries. In these one can retrieve the painting only by its name, which causes problem when you want to find the beautiful painting which you have photographed in the gallery (and immediately forgot the name). This system requires a photograph of a painting on the input and returns the name and the author of the painting found on the photograph. The system works in 2 phases. Firstly, it segments the region consisting of the painting and the frame from the photograph. Segmentation is done by different methods which includes different preprocessing steps, edge enhancement methods followed by the Hough transform [14] or watershed transformation. In the next step the corresponding painting from the database of originals is retrieved. The retrieval is done by comparing the descriptor file of the segmented region created using SIFT [10] or SURF [2] algorithms with the descriptor files of the paintings stored in the database.

This paper is organized in the following way: In the first section the works of other authors in the area of the classification and the digital preservation of art are presented. In the second section the datasets used in this paper for testing and verification are presented. In the third section the process of the segmentation is detailed. In the fourth section the classification methods are recounted. In the fifth section the paper presents the comparison of different methods and the sixth section concludes the paper.

2 Previous work

Since the 80's the computer graphics and vision community is focusing on the problem of the preservation of the cultural heritage. This big mission includes the restoration and the classification of the fine art painting. In this area the most significant assignments are the digital restoration of the paintings, classification of the author's style and categorization of paintings based on the style [8], distinguishing paintings from real scene photographs [5] and determination of new features for paintings classification (Example: Description of painting's texture using brush strokes [15]). For the relatively complete overview see Lombardi [9].

Works in the area of the classification of paintings are mostly focused on author's style or iconography. However one paper shares the assignment with this paper. In the paper [4]a group of students from Stanford were clas-

*zhaladova@gmail.com
†sikudova@sccg.sk

sifying the paintings from photographs they took in the Cantor Arts Center. Photographs were taken under constant lighting, without any distractive elements covering the painting. Under these conditions segmentation of the painting from the photograph could be achieved with simple thresholding method. For the classification they used the matching of the color histograms of the photographs with the database of the color histograms of originals.

3 Datasets

For testing and verification of segmentation and classification methods two different datasets were used. Both datasets consist of images of the paintings created by Rembrandt Harmenszoon van Rijn (Figure 1).

The first dataset, (the Originals), consists of 15 photographs of paintings obtained from the Olga's gallery [12], the internet gallery with over 10.000 works of art. These photographs contain the paintings without a frame or a wall photographs are in the resolution 600 times 600 dpi.

The second dataset, (the Photographs) includes 100 photographs taken in museums or galleries by tourists with unspecified digital cameras. This dataset contains photographs from the collection of the author of this paper, from the initiative on her website [1] and from the travel.webshots [1] web portal. Photographs are in different resolutions, miscellaneous scales and are taken under varying lighting. In 8 images the painting is partly covered by the bodies of tourists.

All of the photographs from both datasets are resized to the width of 600px and converted to gray scale. Conversion to gray scale was done by eliminating the hue and saturation information while retaining the luminance from the HSV representation of RGB values of the image.

4 Segmentation

The goal of the segmentation phase was the segmentation of the painting and its frame in the input image (from the Photographs dataset). Three different techniques were used. The primal one used the Gauss gradient method [7], in the improved method the Anisotropic diffusion [13] was applied and the additional method is based on the watershed transformation [11]. Results of the three different methods of the segmentation are presented in the Conclusion section.

[1] http://members.chello.sk/halada-j/diplomovka.html

Figure 1: Sample images from the Originals dataset (left) and the Photographs dataset (right). Photographs of the Jewish bride painting in the first row, and of the painting: Portrait of a Young Man in the second row.

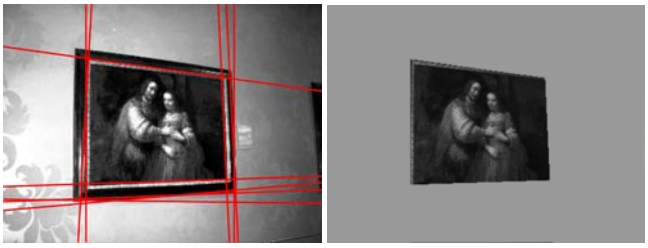
4.1 Gauss gradient method

In the primal method the image is processed using Gauss gradient function which computes the gradient using first order derivative of the Gaussian. It outputs the gradient images Gx and Gy of the input image using convolution with a 2-D Gaussian kernel. In the next phase the Gx and Gy gradient images are send as an input to the Hough transform. The Matlab's implementation of the Hough transform is used, since it enables to count the lines from the Hough peaks directly and to connect or trim them based on their length. Lines created in the previous step are then expanded to the borders of the image and lines with big slope are filtered. Lines are then divided into four groups, one for upper, lower, left and right edges. Consecutively, the painting is segmented as the smallest quadrilateral created from the lines. Gauss gradient method is depicted in the figure 2.

4.2 Anisotropic diffusion method

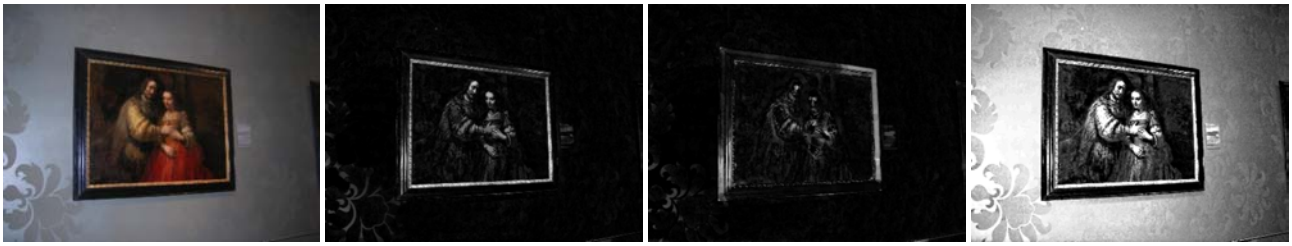
In this approach firstly the histogram equalization is done. Then the image is processed using Anisotropic diffusion, the technique which smooths the image, but preserves the edges. The function is used with the following parameters: number of iterations = 10, kappa = 30, lambda= 0.25 and option = 1 (kappa controls conduction as a function of gradient, lambda controls speed of diffusion, it is 0.25 for maximal stability, option = 1 means the Diffusion equation, this choice favors high contrast edges over low contrast ones). The output image of the function was then convolved with the horizontal and vertical Sobel edge filter, resulting in two binary images Sx, Sy. The images

(a) The input image (b) Gx image (c) Gy image

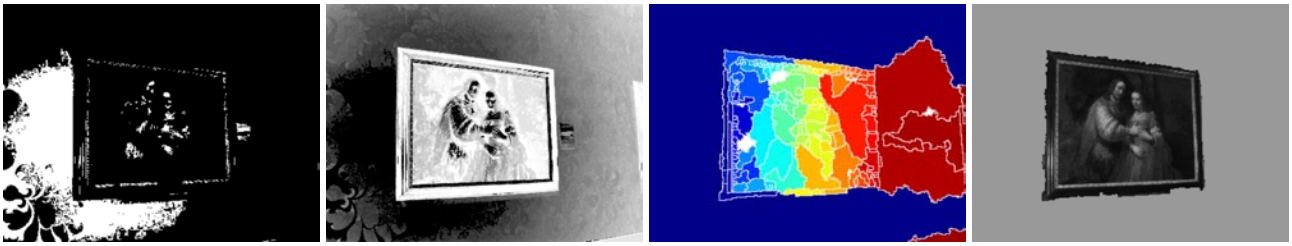


(d) Lines created using Hough transform (e) Final segmentation

Figure 2: The process of the Gauss gradient method of the segmentation.

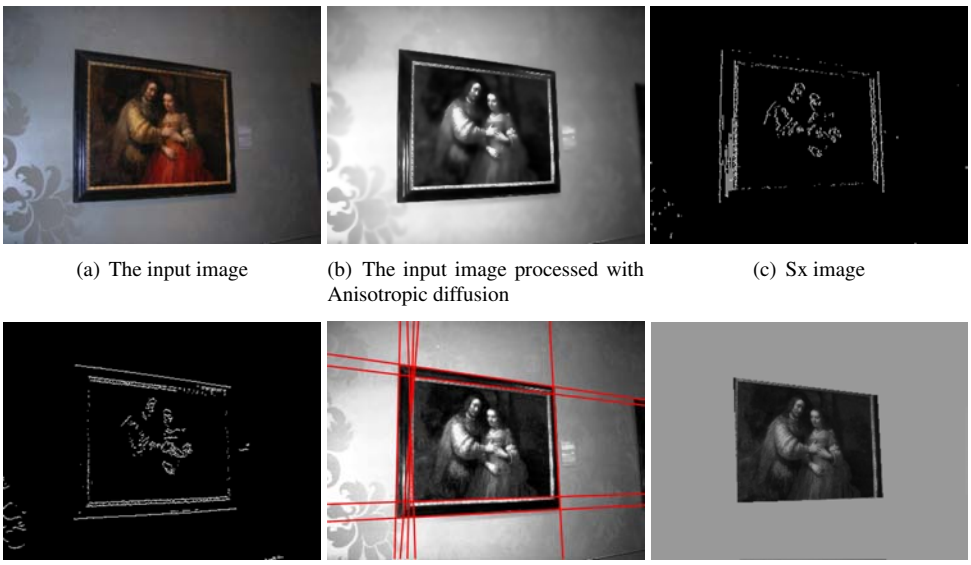


(a) The input image I, (b) Top hat (c) Bottom hat (d) (I+ tophat)-bottomhat



(e) Extended minima of (d) (f) Minima imposition from the complement of (d) with the marker (e) (g) Clusters created with watershed transform (h) Final segmentation

Figure 3: The process of the watershed method of the segmentation.



(a) The input image (b) The input image processed with Anisotropic diffusion (c) Sx image

(d) Sy image (e) Lines created using Hough transform (f) Final segmentation

Figure 4: The process of the Anisotropic diffusion method of the segmentation.

Sx, Sy are processed equally to Gx and Gy images in the first method, and also the input image is segmented in the same way. Anisotropic diffusion method is presented in the figure 4.

4.3 Watershed method

In the third approach the input image is firstly preprocessed to enhance edges of the painting's frame. Afterwards the watershed transform is applied. The preprocessing phase consists of 4 steps: 1. Create top and bottom hat of the input image. 2. Create image I2= (I+ tophat)-bottomhat. 3. Create Im3 as extended minima (regional minima of the H-minima transform) of Im2. 4. Im4 is created as the minima imposition from the complement of Im2 with the marker Im3. In the next step, clusters are created with watershed transform applied on the Im4 image. In the last phase the final segmentation is made by growing the background from the corners in the clustered image. Watershed method is presented in the figure 3.

5 Classification

The process of the classification of the painting segmented from the input image is divided in two steps. First the database of descriptor files of the Originals is created. Then the descriptor file of the painting is matched with the database to find the corresponding original. The descriptor file is a N by M matrix, where N is number of the interest points found in the image and M is the length of the descriptor (128 values for SIFT and 64 for SURF). The two different methods are used for producing the descriptor files. SIFT (Scalable Invariant Feature Transform) developed by D. Lowe [10] and SURF (Speeded Up Robust Features) developed by H. Bay et al. [2].

5.1 SIFT

Sift method consists of a detector and a descriptor of features invariant to translation, rotation, scale, and other imaging parameters.

In the first step of the method the interest points (IPs) are identified in the image by the detector. Then the descriptor for each IP is created. The detector is localising the IPs in the scale-space pyramid, which is created by the consequent scaling of the image, its filtering with the Gaussian kernel and the substraction of subsequent filtered images in each scale. IPs are chosen as the local extremes in the 3x3x3 neighbourhood.

The descriptor for each IP is summarized from the orientation histograms of 4x4 subregions of the IP neighbourhood. In every sample point of the subregion the size and the orientation of the gradient is computed and weigted by the Gaussian window indicated by the overlaid circle. Orientation histogram with 8 directions is created from these values. The descriptor of IP than consists of 8 values for all 16 subregions (128 values).

5.2 SURF

SURF likewise SIFT includes the detector and the descriptor. SURF also operates in the scale-space for identifing the IPs, but unlike SIFT it convolve the original image with the different scales of the box filters (approximations of the Gaussian second order partial derivatives in the y a

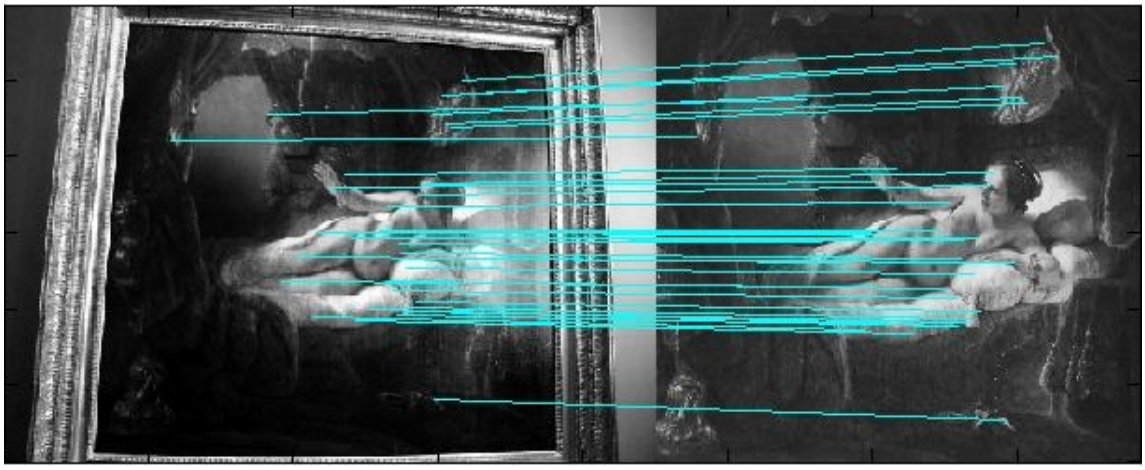


Figure 5: Correspondence of interest points between two paintings matched with SIFT. Painting Danae from the Photographs dataset on the left side and Danae from the Originals on the right side.

xy directions). In order to localise IPs in the scalespaces, a non maximum suppression in a 3x3x3 neighborhood is applied.

64 valued descriptor is created in a few steps. Firstly, the dominant orientation of the IP is extracted from the circular neighborhood as the longest vector estimated by the calculating the sum of all response (the Haar-wavelet responses in the x and y direction, weighted by the Gaussian window) within a sliding orientation window covering the angle of $\frac{\Pi}{3}$. Than, the square region around the IP is created and oriented along the dominant orientation. Lastly, the region is divided into 4x4 subregions. In every subregions 4 features are counted from 5x5 uniformly distributed points. These 4 features are $\sum dx$, $\sum dy$, $\sum |dx|$, $\sum |dy|$: sum of Haar-wavelet responses in the horizontal and vertical direction and the sum of absolute values of Haar-wavelet responses in the horizontal and vertical direction. Four features for all of 16 regions produce the 64 values for every IP.

5.3 Matching

In the matching phase of the classification the binary representations of the descriptor files from the database are loaded and matched with the descriptor file of the segmented painting (DF1) using the nearest neighbor technique. In both SIFT and SURF approaches for each descriptor file (DF2) from the database, the value of the matching with DF1 is counted. For every row (corresponding to the descriptor of one IP) of DF1 the nearest neighbor and the second nearest neighbor from DF2 is counted. Nearest neighbor is a row from DF2 with the smallest Euclidean distance from the DF1 row. The matching value is then the sum of DF1 rows for which the nearest neighbor has value smaller than distanceRatio times second nearest neighbor. The distanceRatio was determined by authors of SIFT and SURF as 0.6 and 0.7. The painting from the Originals dataset with the greatest matching value is elected as the painting best corresponding to the input image.

It is possible that two non corresponding paintings have the matching value greater a 0. This is caused by the fact that features recognized by the SIFT and SURF are not distinctive at 100% and additional inaccuracies are caused by the blur and the noise. In order to prevent incorrect classification of the paintings not present in the Originals database, the threshold for minimal matching value is established. If the DF2 best corresponding to the DF1 of the input painting has the matching value smaller than the treshold, the input painting is not present in the Originals dataset.

5.4 Other methods

In the scope of this paper, one additional method for creating descriptors was verified - the colorSIFT developed by G. J. Burghouts and J. M. Geusebroek [3]. The color extension to the original SIFT considers color gradients, rather than intensity gradients, in the Gaussian derivative framework. This approach, however, proved to be ineffective for our purpose. In the 10 tested images, there was found only 10% of the matches found by SIFT.

6 Results

This section summarizes the results of each step of the proposed system.

6.1 Segmentation results

In the segmentation phase the methods were tested on the Photographs dataset (Images from the Originals dataset are already segmented). This dataset consists of the

Method	Gauss gradient	Anisotr. Diffusion	Watershed
Correct segmentation	73%	89%	49%
Over segmentation	6%	3%	1%
Under segmentation	21%	8%	50%

Table 1: Percentage of paintings properly segmented by different methods

photographs taken by tourists in different galleries, under different lighting condition and with different cameras. Within the segmentation phase most problems were caused by the low contrast of the photographs, which was eliminated in the Anisotropic diffusion method by the equalization of the histogram. Table 1 summarizes the percentage of the correct segmented versus over and under segmented paintings.

Over segmentation, mostly in the Gauss gradient method was induced by the strong edge responds in the paintings, especially in the painting Night Watch (Rijksmuseum, Amsterdam) where the pale flags and spears has very strong color edges in the black background. Other problem with the Night Watch was the low contrast of the black frame of the painting to the dark gray wall paint. In the Watershed method, over segmentation occurs in one image, where the shadow in the upper right side of the image blend with the black upper right corner of the painting. Under segmentation arises, when the paintings frame is mostly covered or in low contrast with the wall or the background of the painting contains strong edges (wall corner or cartouch presented on the photograph).

The problems with over and under segmentation were partly eliminated by using the Anisotropic diffusion in the second method, which smoothes the color edges in the painting and also the edges in the background, but preserve the edges of the frame. The primal method, the Gauss gradient uses smoothing with the Gaussian kernel, which smoothes all edges uniformly. The Watershed method was integrated to present a different approach to the segmentation, but the results indicates that it is not efficient for this purpose. Finally, as expected the best results were achieved with the Anisotropic diffusion method (See table 1).

6.2 Classification results

Two methods were used for the classification purpose, SIFT and SURF. In this section both methods will be evaluated in a sense of a precision and the speed. For the evaluation purpose both datasets were used. One hundred images from the Photographs dataset were divided into 16 groups, 15 groups corresponding to 15 originals present in the Originals dataset and one group with the images of the paintings not presented in the Originals dataset. The descriptor file from segmented paintings was created and the best matching original was chosen. Painting was labeled with the number of the best matching original and the label was compared with the number of the image's group. If the best matching original has matching value smaller then a threshold (the threshold was established on the value 7 for SIFT and 6 for SURF), the image was labeled with the 16- not presented in the database. Correct classification means that image was labeled with the number equal to the number of its group. Table 2 present the number of correctly classified images with the SIFT and the SURF methods. Additional value for the performance measure is the time of the computation of one descriptor file in Matlab. The time value is presented in the table 3.

Method	SIFT	SURF
threshold = 0	75%	73%
threshold = 6	88%	90%
threshold = 8	89%	88%
threshold = 12	90%	82%

Table 2: Percentage of properly classified paintings by SIFT and SURF methods with different thresholds

Method	SIFT	SURF
time of the computation of one descriptor file	0,8125 s	0,32025 s

Table 3: Time spent on the computation of one descriptor file with different methods

7 Conclusion

As a conclusion the best results in the segmentation and the classification of fine art paintings from photographs were achieved with the combination of Anisotropic Diffusion and SURF methods. Both SURF and SIFT achieved 90 % percent of succesfully classified paintings, which means that 90 photographs were correctly named as the name of the painting presented in the photograph. From the 10 uncorrectly classified photographs 5 (3 in SURF) was falsely classified as not present in the database and 5 (7 in SURF) was classified with the name of the wrong painting. Unlike the simmilar classification results, the SUFT method prove to be 2 times faster in the creation of the descriptor files. The best segmentation measure was 89 % and the classification measure was 90 %.

8 Future work

In the next phase of the work the classification process will be improved by adding additional criterion to the matching process. The paintings will be compared also by the aspect ratio. This criterion will be helpful for the images with the best matching value close to the threshold. This value may also accelerate the matching process- the descriptor file of the image will be matched only with DFs of the paintings with the similar ratio values.

9 Acknowledgments

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

References

[1] Inc. AG.com. Travel webshots., 2009. http://travel.webshots.com.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[3] Gertjan J. Burghouts and Jan-Mark Geusebroek. Performance evaluation of local colour invariants. *Comput. Vis. Image Underst.*, 113(1):48–62, 2009.

[4] Etezadi-Amoli M. Chang C. and Hewlett M. A day at the museum., 2009. http://www.stanford.edu/class/ee368/Project07/reports/ee368group06.pdf.

[5] Florin Cutzu, Riad Hammoud, and Alex Leykin. Distinguishing paintings from photographs. *Computer Vision and Image Understanding*, 100(3):249 – 273, 2005.

[6] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):1–60, 2008.

[7] Xiong G. Gradient using first order derivative of gaussian., 2009. http://www.mathworks.com/matlabcentral/fileexchange/8060-gradient-using-first-order-derivative-of-gaussian.

[8] Shuqiang Jiang, Qingming Huang, Qixiang Ye, and Wen Gao. An effective method to detect and categorize digitized traditional chinese paintings. *Pattern Recognition Letters*, 27(7):734 – 746, 2006.

[9] Thomas Edward Lombardi. *The classification of style in fine-art painting*. PhD thesis, New York, NY, USA, 2005. Adviser-Tappert, Charles and Adviser-Cha, Sung-Hyuk.

[10] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2002.

[11] Jiri Militky. Image analysis and matlab., 2009. http://centrum.tul.cz/centrum/itsapt/Summer2005/files/militky_6.pdf.

[12] Mataev Olga. Olga's gallery., 2009. http://www.abcgallery.com/index.html.

[13] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. Technical report, Berkeley, CA, USA, 1988.

[14] Hough P.V.C. Method and means for recognizing complex patterns.

[15] Robert Sablatnig, Paul Kammerer, and Ernestine Zolda. Hierarchical classification of paintings using face- and brush stroke models. In *in Proc. of 14th Int. Conf. on Pattern Recognition*, pages 172–174, 1998.

Rendering

Traversal methods for GPU ray tracing

Marek Vinkler*
Supervised by: doc. Ing. Jiří Sochor, CSc.†

Faculty of Informatics Masaryk University
Brno / Czech Republic

Abstract

Ways of exploiting the raw performance of GPUs for computing ray tracing have been a hot research topic recently. Performance similar to the multi-core CPU ray tracing engines has been achieved. In this paper we present a new traversal method created by combining two of the existing methods. The proposed method is less sensitive to performance loss due to certain object scene distribution. It can also be faster than the methods from which it originated. Several possibilities how to create such a method exist and even better methods can be constructed with the addition of a single instruction to the next generation of GPUs. The resulting ray tracer achieves 50+ fps for primary rays on moderately complex scenes running on current mainstream GPUs.

Keywords: Ray tracing, GPU, CUDA

1 Introduction

The graphics hardware has witnessed enormous growth in both performance and programming flexibility recently [5, 7]. This development enabled creation of general purpose parallel computing architectures such as NVIDIA CUDA [8]. Many applications have been ported to this platform to take advantage of its raw performance. The GPU ray tracing engines became a serious alternative to CPU-based ones. Several mappings of the ray tracing algorithm to the graphics hardware were proposed [4, 1]. In this paper we compare some of the existing methods focusing on their performance characteristics with regard to the object scene distribution. It shows that where one method works well the other often performs poorly. This leads to the idea to create a hybrid method that addresses the drawbacks of both of the existent methods.

There are several promising mappings of the ray tracing algorithm to the graphics hardware. The packet traversal method is described in [4]. In this mapping all rays in a packet follow the same path during the traversal. To leverage the power of the GPU architecture rays are mapped to threads and packet size is chosen as warp (group of parallel threads) size. This way the threads within the warp can cooperate in loading the node and triangle data. Also there is less branching as all the rays, by definition, follow the same code path. This leads to almost perfect utilization of the hardware but effective parallelism decreases each time the threads within the warp want to take different paths.

Another approach named "if-if" traversal can be found in [1]. In this mapping once again one thread computes one ray but these rays follow their individual paths. Thus there is no cooperation among the threads. This approach sacrifices coalesced loads and coherent branching for higher parallelism. It achieves higher performance than packet traversal method when rays take different paths frequently e.g. near the leaf nodes. On the other hand its performance is inferior in places where rays traverse the acceleration data structure coherently e.g. near the tree root.

It is not a coincidence that both of the articles mentioned above use AABB BVH (Axis Aligned Bounding Boxes Bounding Volume Hierarchy [3, 9]) as an acceleration data structure. It has been chosen for several reasons. First, the acceleration data structure should be small as there is considerably less memory available on the GPU [6]. It should also allow fast reconstruction thus supporting dynamic scenes. Both of these criteria favour grids and BVHs. However grids often perform poorly on scenes with non-uniform object scene distribution making BVH the acceleration data structure of choice. Detailed description of both of these traversal methods as well as of the hybrid method constructed from them is given in section 3.

2 Test setup

For the reasons listed above we have chosen AABB BVH as an acceleration data structure for our ray tracing engine. Hierarchy construction is handled by a third-party code from Arauna ray tracer [2]. The maximum number of primitives in a single leaf node is set to 6. The engine creates the hierarchy just once at the beginning allowing

*xvinkl@mail.muni.cz
†sochor@fi.muni.cz

only static geometry to be rendered. However with the change of the underlined BVH construction algorithm dynamic scenes could be rendered as well.

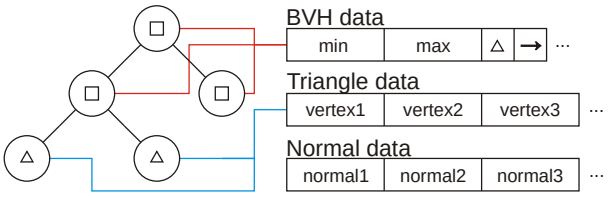


Figure 1: Memory layout

Each node is a 32-byte wide block (12 bytes min coordinates, 12 bytes max coordinates, 4 bytes number of contained primitives and 4 bytes pointer to the children). The latter two are integer values promoted to single precision floating point values for the ease of storing. The left and right BVH children are stored in continuous 64 bytes and are always loaded together. All data are stored in 1D float arrays. There are separate arrays for BVH data, triangle data and vertices normals as shown in 1. The BVH data are always fetched through texture while triangle data use texture only for the packet traversal method. Normal data are always loaded directly from global memory. This way the engine achieves maximum performance.

Four popular test scenes were chosen for performance measurement. Table 1 lists important information about the models. No hand tuning was applied to the assembly code produced by the complier.

Scene	Triangles	Split triangles	Nodes
Conference	283k	424k	190k
Fairy	174k	261k	117k
Sibenik	80k	121k	56k
Sponza	76k	114k	52k

Table 1: Triangle counts, triangle counts after split and number of nodes for the four test scenes. Rendered images of these scenes are in Figure 2.

3 Methods

To gain maximum ray tracing performance from the GPU one needs to address several key aspects. First, the register count per thread must be kept small enough to allow sufficient parallelism. Some thread data (e.g. stack, next node pointer, etc.) must be stored in shared or local memory instead of registers. Second, the code for data loading, stack handling and traversal decisions should be kept as simple as possible not to waste instructions. The kernels are completely compute bound (the memory access latency is completely hidden with computation) and therefore any extra instructions lead to noticeable performance loss.

We have implemented and compared three ray tracing traversal methods: packet traversal, if-if traversal and hybrid traversal. All of these methods share a common pattern and differ only in how the traversal is done. They all use optimization techniques described in [1]. Namely persistent warps and assigning ray indices based on morton code are employed. Description of each traversal method as well as its advantages and disadvantages follows.

3.1 Packet traversal

The basic characteristic of packet traversal is that a group of rays follows exactly the same path in the BVH tree. This is achieved by sharing the traversal stack among the rays in the packet. Each time the rays want to decide which node to traverse next they have to vote. There are two options how to do that using current hardware. The first one is to do reduction in shared memory. The second is to make use of warp vote functions. These functions evaluate a predicate for each thread and then return the same boolean value (computed from those evaluations) to every thread in the warp. Currently vote functions *__any* and *__all* are supported. While the former method gives a precise answer to where the majority of threads wants to go it takes a lot of instructions to compute. The latter method is only approximate but uses just a few instructions leading to higher performance. Another advantage of coherent traversal is that it leads to perfect memory loads. Every node intersected by the packet is loaded from the global memory only once (not multiple times for every ray in the packet) and all loads are coalesced. Also only one (two for triangle data) memory instruction per thread is issued to load all of the data for all of the threads. However when the rays within the packet want to take different paths in the tree these paths must be serialized using the shared stack. Parallelism is therefore lost with each such branching and rays visit (potentially many) nodes which they do not intersect.

The packet traversal method is the fastest possible method when the rays want to take nearly the same path, for example if the first intersection for all rays within the packet lies in the same leaf. This is however seldom even for highly coherent primary rays. To exploit the architecture best the packet size is chosen as the warp size. Our implementation of packet traversal kernel uses 25 registers per thread and about 3kB of shared memory per block. This leads to 50% occupancy on devices of compute capability 1.2 or higher. This is more than sufficient as the kernels have high arithmetic intensity (ratio between arithmetic and memory operations).

3.2 If-if traversal

In contrast to packet traversal in the if-if traversal method each ray follows its own path. This is done by keeping separate stack for each ray. The stack is currently allocated in thread's local memory as shared memory is too small. Even though the local memory is as slow as the global memory, stack loads and stores latencies seem to be hidden by other threads computation. What actually hurts the performance is thread serialization during these memory operations. Different threads often happen to access different stack indices. In such a case separate memory instructions must be issued for each stack index following the coalescing rules. Another performance loss is due to branching within a warp - some threads want to intersect nodes while others want to intersect triangles. In this case both execution paths are serialized as described in [8]. Clearly this increases the number of issued operations and reduces performance. Another drawback of this method is the way how loading of both node and triangle data is handled. Each thread must issue several memory load instructions to gain the data it needs.

The final kernel consumes 24 registers per thread and no shared memory is needed leading to 63% occupancy. Interestingly there is little performance loss if one extra register is consumed and occupancy drops to 50 %. This method's performance is superior to the one of packet traversal if threads within the warp take different paths frequently. A perfect example is the tracing of secondary rays.

3.3 Hybrid traversal

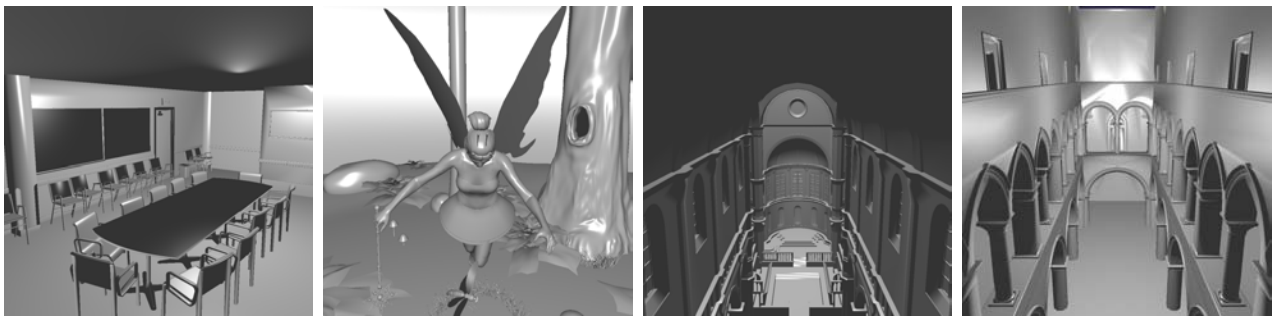
As mentioned above this method is a combination of the packet and if-if traversal ones. The idea is that packet traversal performs best near the tree root where rays are coherent whereas if-if traversal is better suited for traversing nodes near the leaves. It is, however, unclear when and how to switch between the two of the methods. We can divide the methods into groups based on how the switching is done.

The straightforward idea is to switch the packet and if-if traversal each time a certain condition is met. We start tracing the rays with the packet traversal method and when the condition is triggered we switch to if-if traversal. Then after all rays have finished traversing the current path they load another node from the shared stack and start tracing it again with packet traversal. This continues until the shared stack is empty. The traversal thus follows the classical depth-first search scheme but uses different methods to trace different parts of the tree. This method turns out to be slow as the next packet traversal phase cannot start until all of the rays have finished their if-if phase. Thus all the rays are idle until the longest running one ends and this happens multiple times. If, however, there were an effective algorithm for loading work per ray as discussed in [1] the majority of rays would not be idling and the method could be interesting.

Another option is to switch between the packet and if-if traversal only once. This method seems to be more promising and so we have developed several conditions to rule the switching. The easiest one and currently achieving best performance is the one we call "stack-max" traversal method. In this method packet traversal ends when the shared stack size is bigger than a predefined threshold. In this moment if-if traversal starts from the last visited node and later on visits each of the nodes on the shared stack. We will discuss the performance characteristics later in section 4. The register usage for this kernel is 26 registers per thread and the same amount of shared memory as for the packet traversal is used. The occupancy is thus at 50 %.

There is also the possibility of counting how many times rays wanted to take different paths. If this number exceeds some threshold we make the switch. This method is not very sophisticated yet it takes quite a lot of extra instructions leading to poor performance. This is why we will not mention it in the result section.

The last developed method is the most sophisticated one. It stops traversing a path with packet traversal if too few rays want to take that path. If this happens it pushes the address of the last node on that path to the local stack of threads which want to take that path. Then it takes another node from the shared stack and the process continues. This way it traverses all coherent nodes (nodes which a significant amount of rays want to visit) before switching to if-if traversal. Such a traversal no longer follows the classical depth-first search traversal scheme. It resembles the depth-limited search but the limit is different for each branch. The if-if traversal then traverses the sub trees defined by nodes saved on the local stack. The main advantage of this method is that it is not dependent on some predefined threshold and should classify coherent/incoherent parts of the tree well. The drawback of this method is that computing how many rays want to go to the left and right children of the current node is expensive. With current generation of hardware the reduction in shared memory must be employed to obtain such a number. However, if a new instruction - returning the number of threads in a warp satisfying a predicate - is introduced in next generation of GPUs the method could be the fastest hybrid method. In the result section this method is denoted as "cut" traversal method. This method has higher register and shared memory demands. It consumes 28 registers and about 3.4kB of shared memory per block. Nonetheless, that is still low enough to keep the 50% occupancy.



(a) Conference (b) Fairy (c) Sibenik (d) Sponza

Figure 2: Test scenes

4 Results

Images in figure 2 were obtained at output resolution of 1024x1024 using NVIDIA GTX 280. Each FPS count in table 2 is an average over five different viewpoints. These viewpoints were chosen randomly so that most of the scene triangles were visible from them. The timings include the whole kernel execution (ray generation, traversal, shading, etc.).

As it can be seen in the upper part of table 2 the if-if traversal is the second slowest of all methods for the primary rays. This is interesting since it is reported to be faster than packet traversal [1]. The reason for this discrepancy is unknown to us. Possible candidates are a worse BVH tree construction algorithm or simply a poor implementation of the mentioned method. It is important to realize that poor performance of the if-if traversal reflects also in performance numbers for the hybrid methods.

The stack-max traversal is slightly faster than the packet traversal but the speedup is not interesting by itself. The noteworthy thing is that it has much more balanced performance with different types of rays. The method depends on a constant denoting the maximal number of items in the shared stack. Its performance slightly varies with the change of the constant. When rendering the conference scene the constant for the best performance was mostly so high that the if-if part of the traversal was skipped. The important thing to notice here is that even when the stack-max traversal collapses into the packet traversal it is not much slower than the specialized kernel. Only one extra if-statement is evaluated in the main traversal loop. For the other two scenes a lower choice of the constant leads to higher performance. The constant value 11 was chosen for separate measurements because it achieved reasonable performance in all of the test scenes. This number is, however, scene specific and cannot be considered a general guideline. From some special viewpoints the stack-max achieved noteworthy speedups. This encourages the hypothesis that the tree can be divided into parts of coherent and incoherent traversal. Though the stack-max is not the right condition to guide this division.

The cut method is the slowest method for the primary rays. This is mainly because its performance is limited by the cost of its traversal decision code. It is cheaper to intersect one or two nodes than to decide which node to take next. This makes it a poor choice for current generation of hardware. As discussed above there is potential to change this state making it interesting to benchmark.

The results for the secondary rays (reflected and refracted primary rays) are given in the bottom part of table 2. The important thing about these numbers is that the measurement corresponds to tracing twice as many rays than for the primary rays. The if-if traversal method is the fastest for the incoherent secondary rays as predicted. Interestingly the other methods are not lacking far behind.

The stack-max traversal method performs quite well on the secondary rays. However, as discussed above it needs the right constant for the switch criterion to be fast. Here constant value 7 turned out to give reasonable performance. As one might expect this number is lower than for the primary rays because the rays tend to take different paths often. Thus the number of rays within the warp that want to take the same path drops rapidly with increasing depth. Unfortunately no choice of the constant for the stack-max method forces a collapse into the if-if traversal method. The packet part of the traversal always takes place. This explains why the method cannot achieve performance as good as the one of the if-if traversal method for the Sibenik and Sponza scenes. As discussed above the stack-max traversal method is more versatile with regard to object scene distribution. This can be observed from the speedups against packet traversal for the secondary rays.

The packet traversal method is a poor choice for the incoherent secondary rays. The rays traverse a great amount of nodes they do not intersect and have to finish paths that the majority of the rays want to take before taking their own path. This is why it is almost as slow as the cut method.

The results for the cut method show some promise. It is still the slowest of all the compared methods but the dif-

ference is not as abysmal as for the primary rays. This is a sign that the reduction of the amount of needlessly traversed nodes (due to better traversal criteria) outweigh the criteria cost.

5 Conclusions

The hybrid traversal methods presented in this paper are comparative to the fastest known traversal methods. They are able to benefit from coherent traversal of rays while they sustain good performance in incoherent setting as well. This is achieved by utilizing most of the GPU resources. Advantageously optimizations presented in other papers can be used with this traversal method as well.

The performance of the hybrid methods is strongly influenced by the employed switching criterion. The proposed criteria use heuristics to divide the acceleration data structure into coherent and incoherent parts. Better heuristics may be found in the future. They should be fast, consume as little resources as possible and precise in classification of coherent/incoherent nodes. With the progress in graphics hardware some known criteria may become more efficient.

6 Acknowledgments

Jacco Bikker (http://igad.nhtv.nl/∼bikker/) for the Arauna engine. Marko Dabrovic (www.rna.hr) for the Sibenik cathedral model. University of Utah for the Fairy scene. This work was supported by Ministry of Education of The Czech Republic, Contract No. LC06008 and by The Grant Agency of The Czech Republic, Contract No. P202/10/1435.

References

[1] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pages 145–149, New York, NY, USA, 2009. ACM.

[2] Jacco Bikker. Real-time ray tracing through the eyes of a game developer. In *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Andrew S. Glassner, editor. *An introduction to ray tracing*. Academic Press Ltd., London, UK, UK, 1989.

[4] Johannes Günther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Realtime ray tracing on gpu with bvh-based packet traversal. In *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 113–118, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Mark Harris. Many-core gpu computing with nvidia cuda. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 1–1, New York, NY, USA, 2008. ACM.

[6] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Manocha Dinesh. Fast bvh construction on gpus. In *Computer Graphics Forum Volume 28, Issue 2*, pages 375–384. The Eurographics Association and Blackwell Publishing Ltd., 2009.

[7] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28:39–55, 2008.

[8] NVIDIA. *NVIDIA CUDA Programming Guide Version 2.3.*, 2009.

[9] Stefan Popov, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek. Object partitioning considered harmful: space subdivision for bvhs. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pages 15–22, New York, NY, USA, 2009. ACM.

Primary rays								
Method	Conference	Speed-up	Fairy	Speed-up	Sibenik	Speed-up	Sponza	Speed-up
packet	68.12	0%	60.44	0%	49.48	0%	47.46	0%
if-if	54.80	-19.55%	54.54	-9.76%	49.48	0%	47.80	+0.72%
stack-max(11)	67.44	-1.00%	60.64	+0.33%	49.93	+0.91%	47.40	-0.13%
stack-max(best)	67.92	-0.29%	61.20	+1.26%	49.94	+0.93%	47.56	+0.21%
cut	61.98	-9.01%	53.04	-12.24%	44.96	-9.14%	42.28	-10.91%
Secondary rays								
packet	21.22	0%	18.84	0%	14.94	0%	15.26	0%
if-if	21.80	+2.73%	19.50	+3.50%	16.80	+12.45%	19.18	+25.69%
stack-max(7)	21.72	+2.36%	19.98	+6.05%	15.40	+3.08%	16.26	+6.55%
stack-max(best)	21.78	+2.64%	20.04	+6.37%	15.80	+5.76%	17.32	+13.50%
cut	21.66	+2.07%	18.30	-2.87%	13.82	-7.50%	15.00	-1.70%

Table 2: FPS counts from four static scenes averaged over five viewpoints each. Text in parentheses denotes the constant used for that method. The speed-up against packet traversal method is given for each test scene.

Eye Tracking in Virtual Environments: The Study of Possibilities and the Implementation of Gaze-point Dependent Depth of Field

Bartosz Bazyluk*
Supervised by: Anna Tomaszewska

Computer Graphics Group
West Pomeranian University of Technology, Szczecin

Abstract

In this paper we present the application of an eye tracker as an innovative real-time virtual environment interaction device, that enables a new level of control, and forms a base for many realistic sight-dependent visual effects. Current use and main stream of research regarding eye tracking technology aims at marketing and usability testing, as well as help for people who suffer from manual disabilities. The goal of our work is to spread interest in other uses of this advanced and impressive technology. Apart from the discussion about possible uses of eye tracking in the computer-generated worlds, we provide a complete case study of a depth of field post processing effect for real-time graphics, that relates on the user's gaze point.

Keywords: eye tracking, gaze point, depth of field, focus, virtual environment control, real-time visual effects.

Figure 1: A virtual environment with depth of field dependent on user's gaze point.

1 Introduction

In the past few years a constant search for still more attractive and more innovative controllers used in human to virtual environment interaction can be observed. Examples of this trend that gained commercial success are the famous *Nintendo Wii Remote*, the multiple-input sensitive touch screens offered by Apple in its portable devices and by Microsoft in its *Surface* and *PlayStation Eye* from Sony. Each of these tries to take advantage of previously unused aspects of user's natural activity, improving his experience, and absorbing in an absolutely new and often much more engaging way [6].

Our goal is to encourage the discussion about interaction with artificial, computer-generated world that involves the use of an eye-tracker: a device that unleashes great possibilities that reside in the human visual system by constantly observing the user's eyes, and interpreting their movement to provide accurate information on the person's behaviour at a real time basis. The idea of such application of an eye tracker has emerged together with the recent advancement in eye tracking technology, which made it much easier to use and much more comfortable to utilise (see Section 2).

The first and most basic ideas regarding the use of eye tracking in video games, relied mainly on shifting the control of well-known features from other input devices to the eye tracker. These concepts include gaze-based virtual camera rotation and weapon aiming [10, 13]. Such applications however force the player to unnatural behaviour, requiring him to be constantly aware of where he looks and disallowing him to perform actions as simple as scanning the game status display, without invoking unwanted camera movement. The poor reception of these innovations is clearly shown in [13]. The other approach is driven by the idea of eye tracking as a technology broad in its capabilities to such an extent, that it should not just replace any of the well-known functionality. It rather should expand it, or open the door to a brand new, wide range of possibilities for enhancing the user's experience and creating previously impossible effects that simulate the natural world. Identifying ourselves with the latter belief, we took up the challenge of improving the well known depth of field effect, using an eye tracker.

*bartosz@bazyluk.net

Intending to present one
visual effects, we have bui
the artificial depth of field
user's gaze point (see Figu
assumptions. The develop
lem of universal method f
ent commercially available
we have addressed by cre
meant to solve incompatib
oping visual effects conce
providing ready to use valu
supplied by the device itse

Our paper starts with a b
technology and the way th
lowed by the description o
and the visual effect, that
We then provide a short su
eye tracking in virtual envi
depth of field effect depe
algorithm that we have ch
tation.

2 Background

The eye tracking in its ori
gathering and providing th
time data concerning mov
point-of-regard, a point th
the one that the person is l
formation was acquired in
decades. Inaccuracy, cons
often distracting invasiven
of eye tracking only to sc
fields of psychology and
approach, however, lacks
and therefore may be succ
require user's attention una
ities or those, where the u
role – such as video gamin
ments.

Present day devices in
based combined pupil/corn
or more commonly both su
rect, invisible infra-red lig
ance of so called Purkinje i
which is accompanied by
(see Figure 2). Captured
the infra-red spectrum, the
pupil and corneal reflectio
estimate accurately the ga
ers achieve the accuracy o
After having the device c
quires the subject to follow
points, it is possible to c
space gaze point coordinat

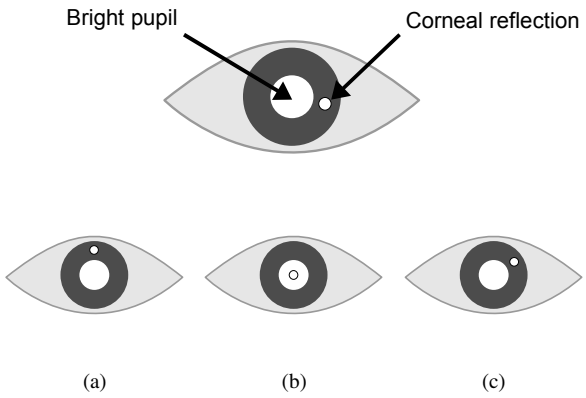


Figure 2: The pupil and corneal reflections as seen by an infra-red camera. Gaze point below the camera (a), in the centre (b) and down and to the right from the camera (c) [14].

2.1 Data provided by an eye tracker

Apart from what seems to be the most interesting, that is position in screen-space coordinates of subject's point-of-regard, eye trackers provide much more useful information that is worth bringing to attention. While calculating the gaze point, an eye tracker utilises many variables acquired from observation of person's eyes actions, that may be employed into calculating visual and behavioural effects. Tracking the movement of both eyes and their Purkinje-image-to-pupil relation, may lead to creation of a head tracking algorithm, and in consequence, the ability to implement extensive number of ways to control the player's avatar in a virtual environment. Also the diameter of subject's pupil, eye's distance from the device, and the camera image may be retrieved [15] depending on the actual device's capabilities.

What is offered by the device's API is not yet everything what may be acquired from an eye tracker. These basic data in connection with the time factor pose a great background to the calculation of more complex, derived values. Beside the most obvious and often necessary fixation times, we propose the introduction of a user's concentration factor concerning an amplitude of his eye movements in certain periods of time. The use of such a variable could involve for example the player's weapon's accuracy.

2.2 Depth of field in real-time graphics

Rendering a scene using artificial methods differs from capturing the real-world scene with a lens equipped camera in the lack of apparatus' physical dimensions. When using an actual device, the rays cast on the sensor are subject to refractions produced by physical properties of the lens. They appear in a photography as the phenomena named circles of confusion (*CoC*s), with diameter varying according to the sensor-lens and lens-object distance. As they blend together, the image appears blurred where the

objects are closer to the camera than it is focused, or when they are too far away. In the case of computer rendering, the image is created with an idealised, pinhole camera containing a virtual lens of zero size. Because of that the rays always cross at a single point, regardless of the distance. To achieve the effect of depth of field, special computation is needed [2].

Depth of field may be called a visual artefact as it limits the acuity of the scene. However, while talking about simulating such visual phenomena by the means of computer graphics, it is important to bring into focus the purpose of that effort. Indisputably the most commonly pursuited branch in artificial generation of images is their realism and effective real-world resemblance. This goal can be divided into three concepts: physical realism, functional realism and photorealism [7]. The latter touches the aspect of viewer being unable to distinguish the computer-rendered image from a photography, what must be achieved not only by accurately recreating the physically-correct light behaviour in the scene, but also by taking into account the limitations of the optics in devices used for image acquisition. Those renderings, which present results impossible to reproduce using conventional photographic techniques, will not fulfil that requirement.

The depth of field rendering is possible using many methods. The one that most closely follows the real-world phenomenon, apart from modelling the light rays behaviour, is an accumulation buffer technique. It involves rendering the whole scene many times to produce one frame, while moving the camera slightly to simulate the acquisition with a device possessing physical dimensions. Despite accurate results, this technique is highly ineffective for real-time graphics due to unacceptable computational overload. There are alternatives that contain simplifications, such as per object multi-layer rendering, or the most common, depth-buffer based methods. The latter owe their popularity to the easiness of obtaining the point's distance to camera, using values existent in the z-buffer. They can be divided into several groups regarding the way of using that information: from forward-mapped scattering methods that are hard to compute parallelly using modern programmable graphics hardware [2], through reverse-mapped gathering methods [5], up to innovative approaches using heat distribution simulation [1].

3 Depth of field with an eye tracker

Having the access to filtered screen-space coordinates of user's point-of-regard, we are able to use the same conventional methods for obtaining the knowledge of place, object, location that the user is actually focused on, as when analysing the mouse pointer coordinates. We present several ideas of utilising these valuable data.

- The ability of calculating the luminosity of an image fragment corresponding to the gaze point and its vicinity, together with high dynamic range rendering allows to alter the scene's exposition level dynamically, which may result in much more realistic tone mapping and accurate blooming, as well as true simulation of human eye's sensory adaptation to varying lighting conditions [11].

- Recording the recent user's scan path can be used to create dynamic, artificial after-images, emphasising the brightness of scene's elements.

- Altering the viewing frustum depending on the user's head position relative to the display screen, e.g. by slightly expanding the field of view when the intermediate distance decreases, creating the impression of looking through the window rather than watching a projected image.

- Also the simulation of vestibo-ocular reflex connected with camera swinging during player's movement may also be a subtle yet heavily immersive addition [9].

These examples of visual effects altering upon gaze shift bring the resulting scene closer to the state of what can be called interactive photorealism.

Worth mentioning is also a completely different application of eye tracking in complex scene rendering, taking into account the inacuity of human peripheral vision in favour of that involving the foveal disk region of retina. Human is in fact able to cover only 1-5 degrees of visual angle with clear, foveal vision, what may be visualised as only 3% of a 21" monitor screen viewed from the distance of 60 cm [3]. What is a human visual system's disadvantage, may be converted into an advantage in the terms of real-time computer graphics, by progressive reduction of rendering quality with the distance from user's gaze point to gain improvement in performance. The quality reduction is possible either in the domain of resolution, displayed scene's geometrical complexity [4], or precision of post-processing effects. This implementation of eye tracking is called the gaze-contingent display method.

In this paper, however, we wanted to present a relatively simple, yet spectacular effect of an artificial depth of field, which varies with the distance from camera of a virtual point that the user is looking at. Our choice emerged according to the fact, that besides improving the impression of scene's depth, an independent of user's eyes behaviour depth of field is often found distracting [8] as it blurs the screen areas that may catch the user's interest. The usefulness of knowledge about the user's gaze point in this field seems to be obvious.

The experiments with interactive, controlled by an eye tracker depth of field effect described in [9] involved the simple reverse-mapped z-buffer method, together with minor artefact correction and a technique of autofocusing, that helped translating the gaze point coordinates provided

by an eye tracker into actual focus distance. It though suffered from a drawback typical to classic depth-buffer approaches that calculate the blurriness with circle of confusion modelling. They are subject to heavily disturbing depth discontinuity problem, when an out-of-focus object occludes the in-focus background. It results in the blurred object's silhouette being clearly visible as a hard edge, which is highly unacceptable [2] (see Figure 4a).

This brought up our deliberations according to the perception of gaze-point dependent depth of field. It is worth noticing, that such way of control results in fully-blurred areas rarely being visible to the user, as they will instantly become clear when gaze is shifted upon them. It is then more important than when creating a non-gaze contingent depth of field effect, to take care of visual artefacts occurring near the objects' edges, as it is where the results of the depth of field will be observed most often. Such artefacts include both depth discontinuity problem and intensity leaking. These observations led us to the search for our own algorithm modification.

3.1 Enhancement of the basic approach

Our approach is an extension to the basic reverse-mapped z-buffer technique with the gathering blurring method based on Poisson disk samples, and involves implementing the extrapolation of *CoC* values for objects closer to the camera than the focus plane. Similarly as in many other methods, we base on a thin-lens camera model and derive from it the circle of confusion diameter equation [5]:

$$CoC = a \cdot \left|\frac{f}{d_0 - f}\right| \cdot \left|1 - \frac{d_0}{d_p}\right| \quad (1)$$

Where a is the aperture, f is the focal length of the lens, d_0 is the focus distance and d_p is the source point's distance to camera. The aperture and focal length need to be selected empirically, as their values should suit the scene's dimensions, the camera's angle of view and the desired intensity of resulting effect. The focal length can be set to increase in very short focus distances, to extend the depth of field for extremely small distance values while maintaining the effect's visibility in longer distances.

To address the depth discontinuity problem described in the previous section, we have surveyed existing solutions. The most basic and popular at the same time [2], involves blurring the *CoC* values represented as a texture, which makes the object edges semi-transparent, as they are mixed with background in the proportion of 50% to 50% (because of blurred *CoC*'s gradual distribution; see Figure 3b). We however decided to follow [5] to calculate extrapolated *CoC* value, using both original CoC_o and blurred CoC_b textures, with the formula:

$$CoC = 2 \cdot max(CoC_o, CoC_b) - CoC_o \quad (2)$$

In effect, we blur the occluding object outside its silhouette, which results in nice, soft edges. The downside is

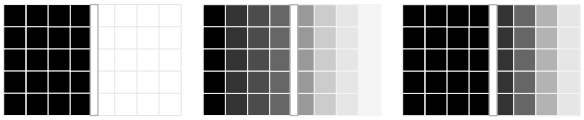


Figure 3: Extrapolation of *CoC* values using original (a) and blurred (b) *CoC* texture comparison to produce the result (c) *CoC* texture. Centre line is the object's silhouette, and darker pixels represent the area with higher *CoC* radius.

the fact of background as well being blurred in the occluding object's vicinity (see Figure 4c). This problem may be addressed with multi-layer rendering or per-pixel layers [12], but it would involve multiple scene rendering what we tried to avoid, as this common problem seems to be far less intrusive than the former hard edges.

Intending to test our implementation with a modern, accurate eye tracker, we have decided to rely on the gaze point data processed in the same way, as for other uses. We have intentionally abandoned creating an algorithm for autofocusing that was proposed in [9], in favour of averaging performed in the interface library.

4 Implementation

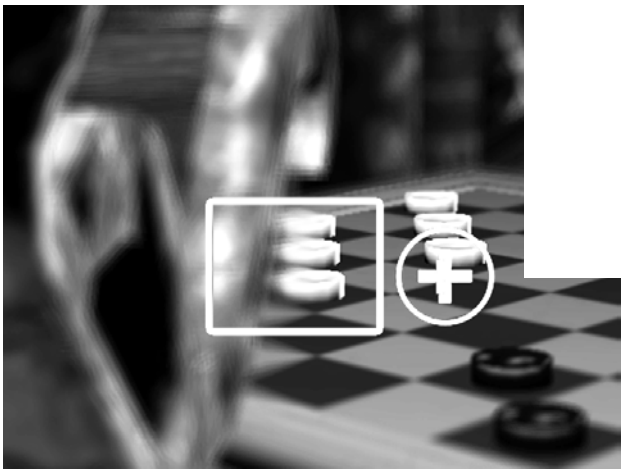
The development of an application that uses the data provided by an eye tracker can be divided into two separate layers: one, that governs the computer-device communication, and the other which is responsible for the data utilisation itself.

4.1 Communication interface

Bearing in mind the diversity of programming interfaces across the available commercial eye-tracking platforms, it is necessary to introduce a solution which will provide standardised output usable in end applications. Our proposition is a universal library, utilising the concept of adaptor-based architecture (see Figure 5). Offering the ease of extending compatibility to new devices, exposing both primitive and derived data in a uniform, platform-independent manner, as well as offering device capability tests, the library may become a major step in popularising the idea of adopting eye-tracking into the field of virtual reality and gaming. At the time this article is being published, the adaptor for SensoMotoric Instruments (SMI) API is under development, and two adaptors useful for application development and debugging are completed: first simulating the eye tracker with a mouse, and second that provides an artificial gaze path at random or constant basis. The latter two were used in the development process of our demo. Further tests and research will involve the SMI RED250 eye tracker. The library is due to be released publicly when completed.

(a)

(b)



(c)

Figure 4: The depth discontinuity problem (a) and the solution (b) with its downside (c).

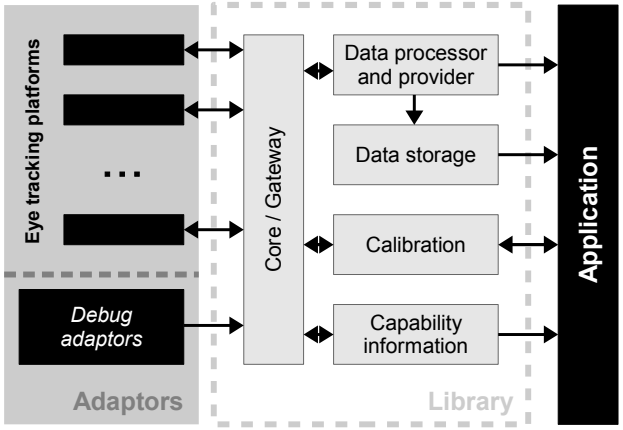


Figure 5: The proposed library architecture.

4.2 Demo application

To test the method, we have prepared a first person perspective demonstration program written in *C#* language, using the most recent *OpenGL 3.2* together with *GLSL 1.50* in its core profile, and utilising the *.Net* based *OpenTK 1.0 beta-2* library. The existing scene is intended to be a testbed for future studies regarding eye tracking applications in both visual effects and interactivity areas. It depicts a fantasy-world interior of a magician's house, which may be easily suited to demonstrate new techniques.

The eye tracker communication layer, utilising its own thread, is the library mentioned in Section 4.1. During each frame generation period, it is queried for both raw and filtered screen-space coordinates of the user's current gaze point. Despite the fact that only the filtered values are necessary for the depth-of-field calculation algorithm, both of them are used for drawing optional on-screen feedback.

The original scene is rendered to a buffer, using the *Frame Buffer Object* with two textures attached: one for colour output, and the other for depth values storage. Then the camera distance from the point equal to filtered gaze-point coordinates is acquired, and assumed to be the focus distance. In the next pass, *CoC* values are calculated with Equation 1 from the depth values in regard to the focus distance, and they are stored in a texture: the nearer-than and farther-than the focus plane values separately. The „near" *CoC* texture is downsampled to 1/8th of its size in each dimension, in order to obtain the blurred *CoC* values used for final computation. In the last pass, blur is applied to the colour texture from the first pass, according to the *CoC* value. To obtain the final *CoC* value which determines the actual blurring radius, the „near" *CoC* value is calculated using Equation 2 and then compared with the corresponding value from the „far" texture. The higher value is used. Finally, the image is rendered on the screen (for the complete diagram see Figure 6).

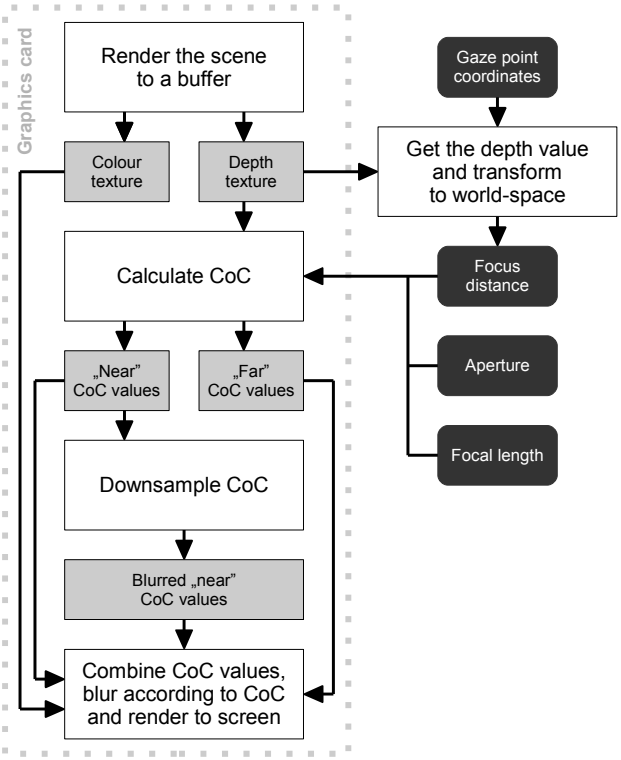


Figure 6: The depth of field rendering algorithm.

5 Results

Developing the demo application enabled us to test our assumptions in practice. The introduction of a consistent interface library led to the possibility of concentrating on the utilisation of provided data itself, rather than processing them. Separating the eye tracker communication from our presentation layer thread resulted in achieving the integration without any noticeable decrease in computing and rendering performance.

The depth of field algorithm that we have chosen to implement delivered convincing simulation of this photographic phenomenon, and offered an improved look of out-of-focus objects occluding the in-focus surface than in the method used in [9]. The implementation did not require us to introduce any changes into the scene rendering process.

The overall outcome brought a new level of interactivity to the artificially generated scene, which proved the very positive results of other studies on using eye tracking in virtual environments [9]. The screens taken from our demonstrative application may be seen in Figure 7.

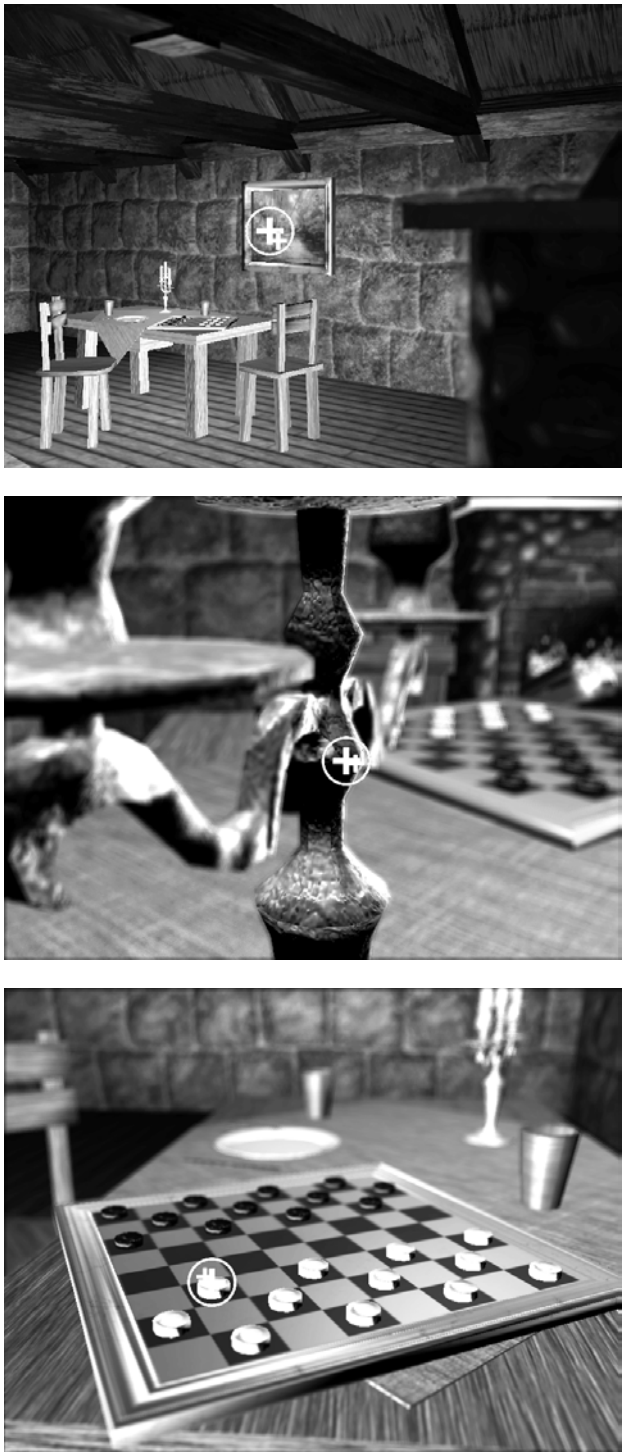


Figure 7: Example screens from our demo, containing on-screen feedback. Green crosshair represents the filtered, while red cross is the raw gaze point.

6 Conclusions and future work

With our work on the demonstration we have proved that modern commercial eye trackers may be successfully used to provide data necessary for rendering advanced, gaze-

aware visual effects in real time. The depth of field algorithm we have used may be improved by addressing the visual artefacts it still produces, like intensity leaking or background blurring in the out-of-focus object's vicinity. However, our approach already emerged to deliver satisfactionary and realistic results.

Our future efforts will be aimed at experimenting with other types of visual effects as well, together with eye-based environment controlling and gaze-contingent rendering performance optimisation. Our demo is planned to include a multi-display feature, that will allow observation of actions performed by the user in real time, displaying visible overlays providing statistical data regarding his eyes' current and recent behaviour, without distracting the subject. The ability to save such data for future interpretation is also projected.

Being aware that eye tracking is consequently gaining interest in the entertainment and portable computers sector, it seems correct to assume that during the few upcoming years we should encounter the introduction of such devices to the consumer market. The rumours of Apple trying to implement an eye tracker in the recently unveiled iPad resulted in discussions about the idea of an eye-controlled operating system, and brought into light the patent for gaze vector navigation that the company has been pending [16]. This makes the search for possible eye tracking use in popular, consumer applications very up-to-date and encourages studies in this yet largely unexplored field.

Acknowledgments

We would like to thank Karolina Lubiszewska for her work on 3D models which we have used in our demo.

References

[1] Marcelo Bertalmio, Pere Fort, and Daniel Sánchez-Crespo. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 767–773, Washington, 2004. IEEE.

[2] Joe Demers. Depth of field: A survey of techniques. In Randima Fernando, editor, *GPU Gems*. NVIDIA Corporation, 2004.

[3] Andrew T. Duchowski. *Eye Tracking Methodology: Theory and Practice 2nd Edition*. Springer, London, 2007.

[4] Andrew T. Duchowski, Nathan Cournia, and Hunter Murphy. Gaze-contingent displays: Review and current trends. In *Adaptive Displays Conference, 2004*, Los Angeles, 2004.

[5] Jr. Earl Hammon. Practical post-process depth of field. In Hubert Nguyen, editor, *GPU Gems 3*. NVIDIA Corporation, 2008.

[6] Ross Eldridge and Heiko Rudolph. Stereo vision for unrestricted human-computer interaction. In Asim Bhatti, editor, *Stereo Vision*. InTech, Vienna, 2008.

[7] James A. Ferwerda. Three varieties of realism in computer graphics. In *SPIE Human Vision and Electronic Imaging 2003*, Bellingham, 2003. SPIE.

[8] Sebastien Hillaire, Anatole Lecuyer, Remi Cozot, and Gery Casiez. Depth-of-field blur effects for first-person navigation in virtual environments. In *IEEE Computer Graphics and Applications*, pages 47–55. IEEE, Los Alamitos, 2008.

[9] Sebastien Hillaire, Anatole Lecuyer, Remi Cozot, and Gery Casiez. Using an eye-tracking system to improve camera motions and depth-of-field blur effects in virtual environments. In *IEEE Virtual Reality Conference 2008*, pages 47–51. IEEE, 2008.

[10] Erika Jönsson. If looks could kill – an evaluation of eye tracking in computer games. Master's thesis, Royal Institute of Technology, Stockholm, 2005.

[11] Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Perceptual effects in real-time tone mapping. In *Spring Conference on Computer Graphics, 2005*, 2005.

[12] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using point splatting on per-pixel layers. *In Computer Graphics Forum*, Vol. 27 Issue 7:1955–1962, 2008.

[13] J. Leyba and J. Malcolm. Eye tracking as an aiming device in a computer game. Technical report, Clemson University, Clemson.

[14] Alex Poole and Linden J. Ball. Eye tracking in human-computer interaction and usability research: Current status and future prospects. In C. Ghaoui, editor, *Encyclopedia of Human-Computer Interaction*. Idea Group, Inc., Pennsylvania, 2005.

[15] SensoMotoric Instruments GmbH. *RED250 Technical Specification*, 2009.

[16] Chris Stevens. *Is Apple about to open a can of eye-tracking?*, January 2010. http://recombu.com/news/a_M11321.html.

[17] Tobii Technology AB. *Tobii T/X series Eye Trackers. Product Description*, 2.0 edition, 2009.

Real-Time Global Illumination in Point Clouds

Reinhold Preiner*
Supervised by: Michael Wimmer†

Institute of Computer Graphics And Algorithms
Vienna University of Technology
Vienna / Austria

Abstract

In this paper we present a real-time global illumination approach for illuminating scenes containing huge point clouds. Our GI approach is based upon the distribution of Virtual Point Lights (VPLs) in the scene, which are then used for the indirect illumination of the scene geometry, using Imperfect Shadow Maps for visibility calculation of the VPLs. We are able to render multiple indirect light bounces at real-time rates, where each light bounce handles the transport of both the diffuse and the specular fraction of the reflected light.

Keywords: Global Illumination, Point Clouds, ISM, Imperfect Shadow Maps, VPL, Virtual Point Light

1 Introduction

Point clouds are a convenient type of geometry representation when huge amounts of geometrical data are to be displayed quickly, or when geometrical data is given in this way (e.g. gathered from a 3D scanning device) and has to be displayed immediately and without a time-consuming triangulation preprocessing step. The Scanopy application[1] developed at the Vienna University Of Technology and at Imagination[2] in Vienna is able to show huge point clouds in real-time and was originally developed to display scanned objects.

There are various approaches for global illumination (GI) in scenes containing conventional mesh geometry. Virtual Point Lights (VPLs) were introduced by Keller in 1997 [4] for indirect illumination, and Ritschel et al [7] proposed Imperfect Shadow Maps (ISMs) as an efficient way for visibility calculation even for a high number of VPLs. The methods and algorithms in our approach base on the thesis of Knecht [6], who implemented GI using VPLs for indirect illumination and ISMs for visibility.[3]

In order to render the ISMs, a number of sample points covering the scene's surfaces, are taken from the meshes, and then are splatted onto the maps. Since we already deal with point clouds describing our geometry, it is an obvious step to take advantage of this fact by applying this GI method on point clouds.

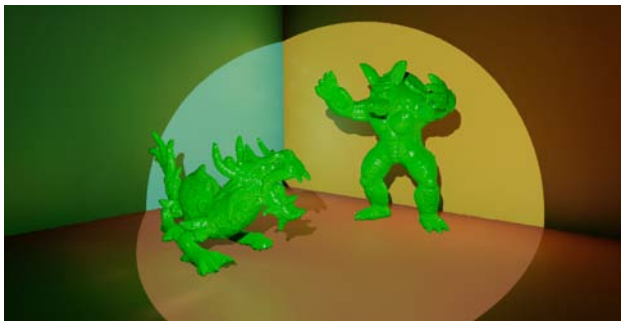


Figure 1: Point cloud scene containing over 5 million points rendered with our global illumination algorithm

Contribution. In this paper, we present our GI-algorithm on point clouds and introduce our approach in illumination computation over several light bounces. Our approach combines the advantages of fast point rendering and efficient ISM rendering for visibility in GI. Further, we advance the illumination method implemented by Knecht by increasing the number of VPLs available for indirect illumination in multiple bounces by simultaneously incorporating one direct- and one indirect bounce with one VPL.

In Section 2 we give a short overview of related work and the background in point rendering, Virtual Point Lights and Imperfect Shadow Maps. In Section 3 and 4, we introduce our approach and describe our algorithm in detail, showing how the implementation of Knecht [6] was advanced in our approach in order to implement real-time GI for point clouds. Finally, in Sections 5 and 6, we present the results of our work and subsequently discuss its restrictions and future work to be done.

2 Related Work

Considering a correct simulation of global illumination in a scene, the rendering equation, first introduced by Kajiya [3] in 1986, represents an ideal and complete description of the illumination process.

*reinholdpreiner@gmx.at
†wimmer@cg.tuwien.ac.at

[1] www.cg.tuwien.ac.at/research/projects/Scanopy
[2] www.imagination.at
[3] Note: Knecht also applies temporal coherency, which we do not.

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega \rho(p,\omega_i,\omega_o) L_i(p,\omega_i) \cos\theta d\omega_i \quad (1)$$

Equation 1 shows a common notation of the rendering equation. The energy radiating from a point p to a direction ω_o equals the sum of energy emitted from p in that direction (represented by emittance-term $L_e(p,\omega_o)$), and the energy from all light from the whole hemisphere above p that is reflected in direction ω_o. The latter is given by the BRDF $\rho(p,\omega_i,\omega_o)$, and the integrand of the incoming light direction $L_i(p,\omega_i)$ incorporates all light incident from the hemisphere Ω over p.[4] θ is the angle between incident light direction and the surface normal at p, and $\cos\theta$ represents a geometry factor that influences the amount of reflected light based on the incident light direction.

Several offline techniques exist, that simulate light propagation as described by the rendering equation, but, since too time-consuming, are inapplicable for real-time rendering. Real-time Global Illumination is a challenging task and with current hardware can often only be achieved by introducing speed-gaining approximations that result in acceptable images.

Keller introduced Virtual Point Lights (VPLs) in his Instant Radiosity approach in 1997 [4]. Those VPLs are seeded over the scene geometry and act as "virtual" light sources for the overall illumination computation of a screen pixel. They represent a sampled subset of all points contributing incident energy on a given point p in the rendering equation. The main issue with the use of many VPLs is the visibility information for all possible outgoing light directions of each single VPL.

In 2008, Ritschel et al [7] proposed so called Imperfect Shadow Maps (ISMs) as a method for efficiently computing indirect illumination. The main idea behind ISMs is, that it is sufficient to create fast and inaccurate (imperfect) shadow maps, if used for a large number of Virtual Point Lights. With growing number of VPLs used for indirect illumination, visible errors or artifacts from the shadow maps' imperfectness get more and more obliterated.

We apply global illumination on point clouds, implemented in a renderer that is able to render enormous point clouds at interactive frame-rates. The technique bases on Wimmer and Scheiblauer's Instant Points approach [8], introduced in 2006. They use a new out-of-core algorithm that uses nested octrees for efficiently building a hierarchy on the point set, significantly reducing the memory-overhead for the data-structure. Figure 2 shows a 3d-scanned point cloud scene rendered in the Scanopy application.

[4]Note: When rendering translucent objects, we have to extend to BSDFs and integrate over the whole sphere around p

Figure 2: Point cloud scene of St. Stephan's Cathedral in Vienna, rendering over 20 million points (over 420 million points in model) with the Instant Points approach in Scanopy (no illumination computation).

3 Overview

Our scene is represented by a point cloud and illuminated by a spot-lightsource. Illumination of the scene geometry mainly consists of two parts: direct and indirect illumination. The direct illumination part is straightforward: The geometry within the spotlight-cone is shaded, and shadow-mapping is performed. Indirect illumination is the more sophisticated part. The term indirect illumination describes all illumination from light rays, which do not come directly from the light source, but rather come from some surface point where the ray was reflected (bounced). Of course, light rays in real world can be bounced several times until their energy is totally absorbed by the reflecting surfaces. Therefore, a good GI implementation also includes multiple light bounces (while maintaining an accordant high frame rate).

To perform indirect illumination, the bulk of light rays from the light source that are reflected at different surface points and from there are illuminating other parts of a scene, is approximated by a number of Virtual Point Lights (VPLs). Those VPLs are seeded over the directly illuminated area in the scene, and from there on each one acts as a point light illuminating the geometry within the whole hemisphere over that point. In an own render pass, those parts of the scene which are visible to the camera are shaded with respect to the previously seeded VPLs. This shading is performed in image space. We use a Camera G-Buffer which stores necessary data of the scene per pixel - i.e. only for those points of the scene which are visible to the camera - and each consecutive shading is performed per pixel on this G-Buffer.

When rendering multiple light bounces, those steps are repeated, i.e. first the VPLs have to be redistributed (originating from the current VPL positions), and then the G-Buffer is again shaded from the new VPL positions, accumulating illumination in the G-Buffer over several bounces.

In order to improve speed, we perform interleaved shad-

ing on the accumulation buffer, as proposed by [5]. Only a interleaved subset of pixels is shaded per VPL, resulting in fewer computations, at costs of reduced quality. After all indirect illumination computations - probably iterative over multiple light bounces - are finished, the interleaved accumulation buffer is merged to an image of original size. This merged image can show local differences in the shading of the pixels (see Figure 7). Therefore, we apply an geometry-aware filter kernel in order to smooth the result image [5]. Finally, direct illumination with shadow-mapping is added, and the resulting HDR scene image is tone-mapped in order to obtain an LDR image of the scene.

Basically, the implementation of indirect illumination in our approach represents an extension to the method described by [6]. While [6] just follows an indirect light ray over several bounces reducing its energy with each reflection, this implementation additionally takes a possible direct illumination of each single reflecting point into account. A comparison of these approaches is given in figure 3.

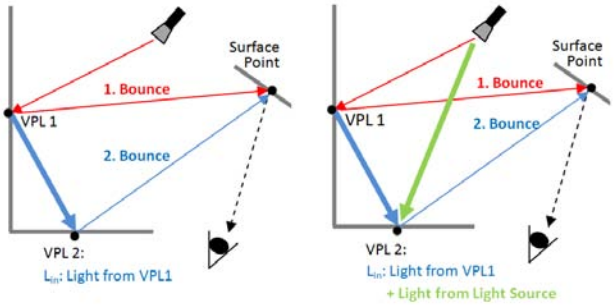


Figure 3: Comparison between Indirect Illumination by Knecht (left side) and our implementation (right side). At multiple bounces, Knecht only reflects the light coming from the last VPL at following VPLs in direction of the surface point to be shaded. The new approach implemented in Scanopy reflects both the light from the last VPL and additionally the possibly incident light coming directly from the light source.

Specular bounces. We are able to correctly render multiple specular bounces. At each bounce, we incorporate both diffuse and specular components of light incident when shading specular reflections[5], considering that the color of specular reflected light deflects over several bounces due to mixed diffuse light (see Figure 4). To accomplish this, for each VPL at each bounce, we cache the incoming light from the previous VPL scaled by the cosine of the light incident angle (geometry term) to lookup the diffuse contribution of any incident light ray.

[5]Note that this is the same as for diffuse reflections

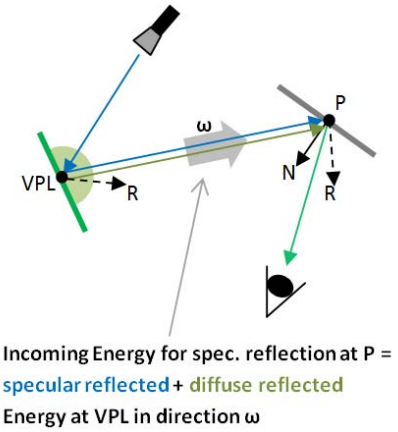


Figure 4: Illustration of our method for computing specular reflections. At each reflection point, N is the surface normal, and R is the direction of max. specular reflection. The incident light reflected at P is the sum of diffuse reflected light (green) and the specular reflected light (blue) at a VPL, resulting in a slightly changed color of the incident light reflected at P.

4 Detailed Algorithm

To perform global illumination in point clouds, our algorithm incorporates several rendering passes, which are illustrated in Figure 5. This sections describes the several steps of the algorithm in more detail. Figure 7 illustrates the intermediate buffers (G-Buffers, ISMs, Accumulation Buffer), and their place in the rendering chain.

4.1 Camera- and Light-G-Buffer

In the first step, the whole scene is rendered from the camera's point of view into a Camera G-Buffer. The G-Buffer stores necessary information of the accordant geometry or surface behind each pixel in image space. This information is distributed over several textures, and consists of the RGB color of the surface, its material properties (diffuse intensity, specular intensity and shininess), the surface normal and the linear depth of the geometry in view-space.

In the second pass, the whole scene is rendered again, but this time from the spot-light-source's point of view. The scene information rendered in this pass is stored into the Light G-Buffer, which contains the same per-pixel information as the Camera G-Buffer, but further stores an importance value. This value correlates with the intensity of the surface color and the surface's shininess (specular power), and is needed for importance sampling of when distributing VPLs in a following step.

4.2 Distribute VPLs

As mentioned before, indirect illumination is calculated by shading scene geometry with respect to Virtual Point

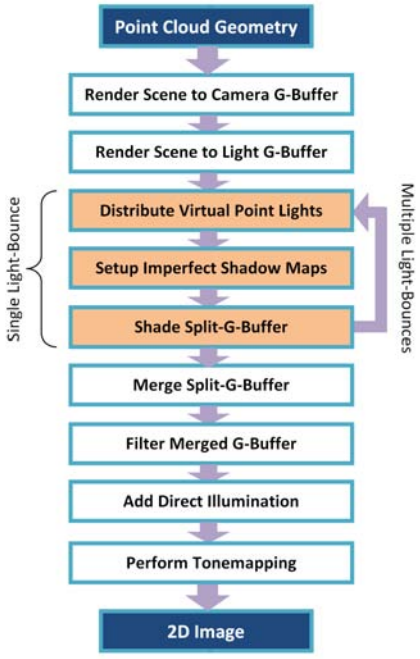


Figure 5: algorithm overview of the global illumination rendering chain

Lights (VPLs), which represent a number of surface points that are directly (or already indirectly) illuminated from some previous light source, and from there illuminate other parts of the scene by reflecting their incident light.

The first set of VPLs is distributed on the surface area illuminated directly by the spot light. The quality of the resulting image increases with the number of VPLs used. However, the number of VPLs significantly influences the frame rate. Therefore, an importance sampling approach is used in order to achieve good results even with fewer VPLs. We start at a pseudo-random distribution of the VPL positions over the spot-light-illuminated area stored in the Light G-Buffer. A 2d-Halton distribution is used in order to achieve a controlled homogeneous distribution, since simple random distributions contain a higher level of noise and locally inhomogeneous areas. Based upon this distribution, hierarchical warping proposed by [2] is used to relocate the VPL positions to obtain denser VPL distributions where needed, and less VPLs where they do not contribute much to the final scene illumination anyway.

When sampling of the VPL positions is done, all necessary data of the VPLs is stored into a VPL-Buffer, which is represented by several 1d-textures. This VPL data consists of the VPL's world space position, the surface normal and material properties (color, reflection indicators) of the surface point the VPL is located on, and two values indicating (or related to) the illuminance of the VPL from both its previous VPL (from a previous bounce) and directly from the light source (see figure 3). The latter values are needed later when calculating the VPL's diffuse reflection of light from both the previous VPL and the light source itself. Since both the sampling of the VPL positions and the hierarchical warping can be done in image space of the light source, all other VPL data can simply be looked up in the Light G-Buffer.

4.3 Setup Imperfect Shadow Maps

The main issue for indirect illuminating of scene points is VPL-visibility. We have to know, whether a currently shaded surface point is visible to a VPL or not. If the point is occluded by some other object, light reflected at the VPL doesn't reach this point and no indirect illumination takes place. Such indirect shadows have a high contribution to scene realism. Figure 6 shows a simple scene setup demonstrating shadows cast from an indirectly illuminated object.

To store the necessary visibility information for each VPL, simple shadow-mapping is insufficient, since the area of the scene visible to a VPL covers the whole hemisphere over the VPL's surface point. Therefore, parabolic maps introduced by Brabec et al [1] are used.

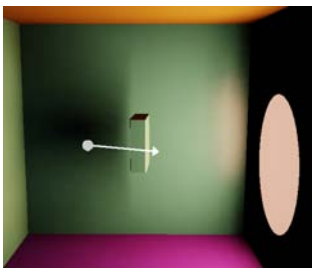


Figure 6: Indirect shadow realized by Imperfect Shadow Maps. The white sphere and arrow indicate the position and direction of the spot-lightsource. Only the wall on the right side is directly illuminated, resulting in the centered cube to cast a smooth indirect shadow.

For each VPL, we create an Imperfect Shadow Map (ISM), as proposed by [7]. To create an ISM, the parabolic projected points of the scene geometry are (box-)splatted onto a map (e.g. by rendering point sprites). The size of a box splat is thereby quadratic proportional to the depth of the point. This imperfect approximation of a perfect shadow map is sufficient for our needs and rendering is much faster this way, especially when using many VPLs. To support large numbers of VPLs, we use one huge ISM buffer which contains all ISMs for each VPL in the scene. Figure 7 illustrates a part of a huge ISM buffer (upper right side of the figure).

To be able to create all ISMs for the whole set of VPLs in the scene in one render pass, we do not render all points of the scene into each single ISM. Rather the whole scene is rendered once, and the points of the scene geometry are distributed over the ISMs as they are passed through the vertex shader. Therefore, each ISM contains only of a subset of the total number of points within the scene, and each point is rendered to only one ISM in fact. This

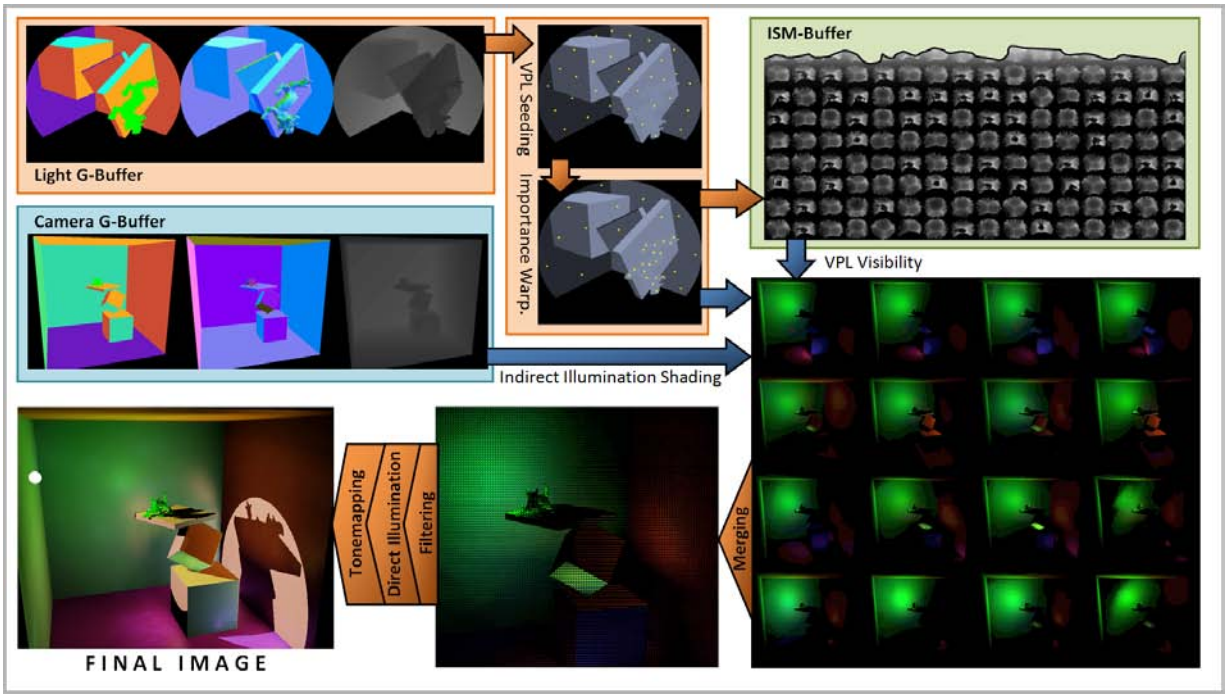


Figure 7: Overview of the GI image synthesis chain and its buffers involved.

further approximation is sufficient for the creation of an ISM, as long as a point cloud of a model within the scene doesn't consist of too few points and the relative number of VPLs is not too high. This point distribution approach makes the time consumed by the ISM setup pass independent of the number of VPLs, which allows for high image quality (high number of VPLs) at real-time frame rates. However, for models with a too low number of points, this distribution approach could certainly yield to unusable, perforated mappings within the ISMs. This problem could be addressed by e.g. rendering multiple passes on the ISM creation stage (using different point-to-ISM assignment offsets per pass), or finding a convenient ISM splat-size scaling factor per point cloud depending on the number of points per ISM and the distance to the ISM's VPL-position.

Since ISMs are created by box-splatting a subset of the points in the scene, the resulting parabolic image can contain holes. To improve the quality of the ISMs, we perform a pull-push-operation on the ISM buffer in order to fill those holes, as described by [6]. The quality of the result depends on the number of pull-push-iterations, but already a number 2-3 iterations can gain good improvements (see Figure 8).

4.4 Shading Split-G-Buffer

Based upon the visibility information encoded for each VPL in the ISM buffer, indirect illumination shading can be performed. As already mentioned, shading of indirect (and, also of direct) illumination is performed in image space, i.e. only for the pixels in the camera G-buffer. In general, for each pixel we would have to calculate the transported energy from each single VPL to the surface point corresponding with that pixel. Since this can be very time consuming with a large number of VPLs, interleaved sampling introduced by [5] is used.

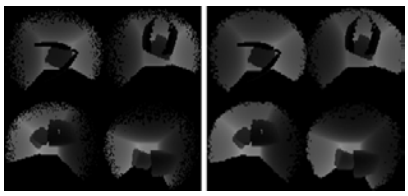


Figure 8: Comparison between a raw ISM buffer (left) and a improved ISM buffer after a pull-push-operation with 2 iterations (right).

4.5 Redistributing VPLs

When rendering multiple indirect light bounces, the VPLs have to be redistributed after each indirect illumination shading pass, in order to obtain a new set of VPLs representing surface reflection points for the next light bounce.

Similar to the method implemented by [6], we assign one new VPL to each current VPL, so that the number VPLs in the scene remains constant. All points of the scene are passed through a shader which distributes all points among the current VPLs and renders a VPL buffer containing the new set of VPLs. To select the best candidate for a new VPL per existing VPL, our shader assigns

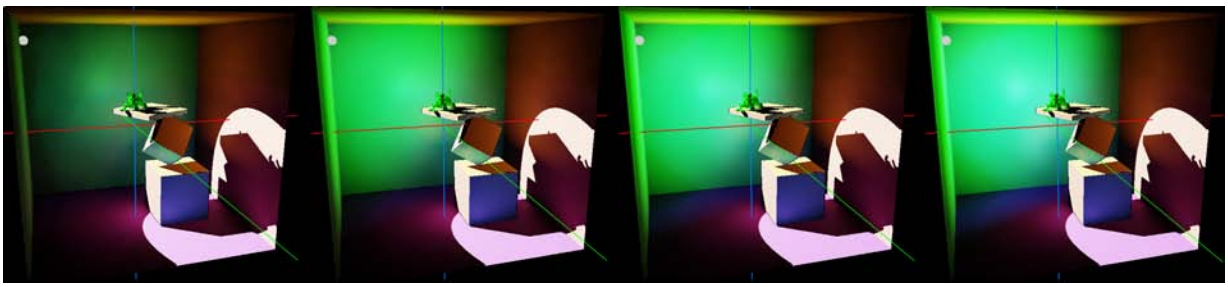


Figure 9: An object stack illuminated by a spot-light inside a Cornell Box, rendered with 256 VPLs and (from left to right) with 1, 2, 3 and 4 light bounces. Considering a distortion by the tone-mapping operator, we can clearly see that the biggest difference in the light situation is given between the first and the second bounce, while the third and fourth bounce only contributes minimal additional brightness.

to each incoming point a z-value that corresponds with the VPL quality of that point. This value is dependent on whether the new point is visible to the current VPL at all, and on how much luminance this new VPL can contribute to the result in the next bounce. Using the right setup, for each current VPL the GPU depth-test automatically leaves those scene points in the buffer, which are best suited as new VPLs. This buffer is then used as new VPL buffer for the next light bounce.

5 Results

Our algorithm supports multiple indirect light bounces at interactive frame-rates. However, in our test scenes, rendering 2-3 bounces already covered the dominant part of the global scene illumination (refer to Figures 9 and 12).

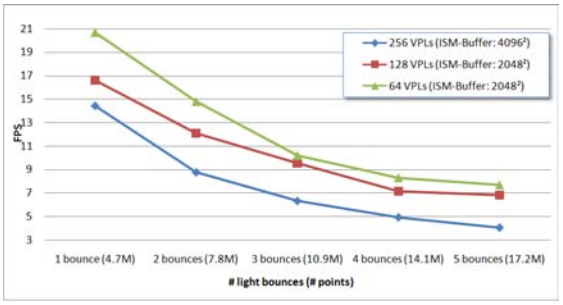


Figure 10: Frame-rate dependency from the number of light-bounces and the number of VPLs in the scene shown in Figure 9.

Figure 9 compares several light bounces in a small Cornell-Box point cloud scene (4.7M points at 1 bounce and 256 VPLs) containing a few boxes and an asian dragon model. Figure 10 illustrates the dependency of the frame-rate from the number of light bounces, the number of VPLs and the size of the ISM-buffer used in this scene. At 128 and 64 VPLs, we used a smaller ISM buffer size (with a different resolution per ISM). Note that with increasing number of light bounces, the total number of points per frame rendered increases too.

Figure 12 shows our GI render mode in the scene of the scanned-dataset of St. Stephan's Cathedral shown in Figure 2, comparing 1, 2 and 3 light bounces at both 4x4 interleaved and non-interleaved VPL shading.[6] The scene was rendered using 256 VPLs. Performance is strongly depending on the number of indirect light bounces, image quality (number of interleaving sub-panes), number of VPLs a.s.o. Figure 11 compares the frame-rates achieved for the St. Stephan's Cathedral scene in Figure 12, rendering 20.6 million points at one light bounce (increases with additional bounces).

The frame-rates shown for both the Cornell-Box and the Cathedral scene were achieved on a platform with a Intel Xeon X5550 2.67GHz CPU, with 72 GB RAM and a GeForce GTX 285 GPU with 2.8 GB RAM.

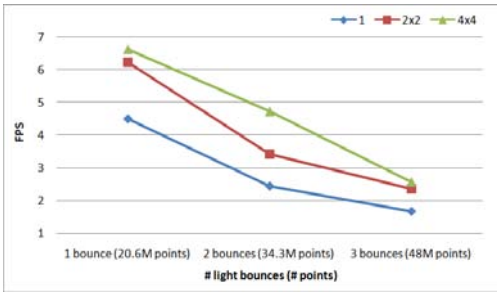


Figure 11: Comparison of frame-rates with 1, 2 and 3 light bounces using 1x1, 2x2 and 4x4 subdivisions for interleaved sampling for the scene shown in Figure 12 (20.6M points).

Due to the importance sampling of the VPL positions by hierarchical warping (concentrating more VPLs on shiny surfaces), we are also able to reproduce caustics from curved surfaces (as shown in figure 13), without the use of additional VPLs. Further, the method of hierarchically warping the VPLs works against a weakness of Virtual Point Lights: the tendency to create sparkles when

[6]Note: The holes in the scene due to an imperfect/incomplete dataset, which was acquired by 3D-scanning. We used an imperfect normal-estimation algorithm, occasionally leading to wrong or missing normals and thus to to local illumination errors.

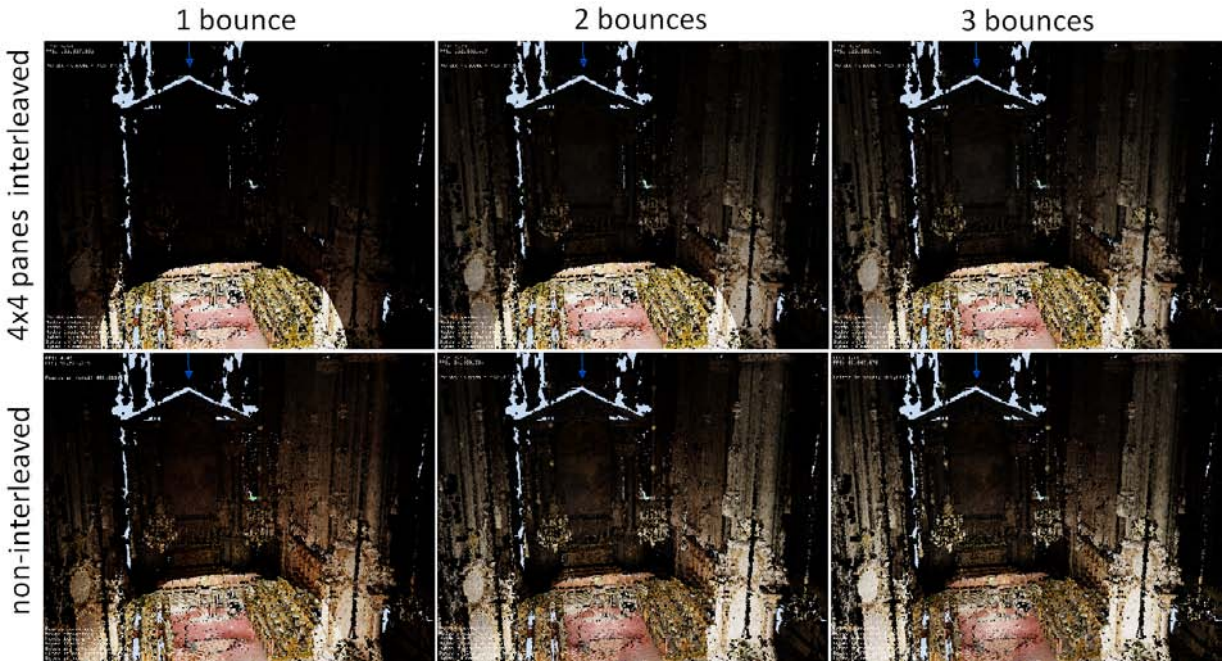


Figure 12: Comparison of different GI-parameters rendering the scanned inside of St. Stephan's Cathedral in Vienna. In the dark scene, we placed a big spotlight at the ceiling pointing at the floor. The upper row shows screens with 4x4 panes interleaved indirect illumination shading. The series in the lower row is rendered without interleaving (consuming more time), which leads to a higher overall-illumination, since each pixel is illuminated by each VPL. Note that with increasing number of indirect light bounces, the higher areas of the walls get more and more illuminated (converging at 3-4 bounces).

placed on surfaces with a high specular component (see figure 14). The appearance of those sparkles often due to an undersampling of VPLs in highly shiny areas, which can not be avoided in cases of too few VPLs in a scene with balanced reflection behavior.

We support transport of specular reflected light over several bounces, allowing for highly specular scenes. However, those situations often require a higher VPL density to avoid the mentioned sparkle effects, while a proper diffuse illumination suffices less VPLs.

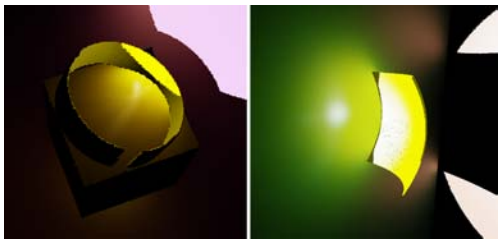


Figure 13: Left: caustic created by a ring. Right: Illumination of a parabolic surface, causing a highlight at its focal point at a nearby wall.

6 Conclusions and Future Work

We have shown a way to perform global illumination on point cloud scenes at real-time frame-rates. It benefits

Figure 14: Scene rendered with global diffuse intensity 0.5, specular intensity 0.95 and shininess 1000. At the highly glossy surfaces, the distribution of VPLs lead to the appearance of sparkles.

from the efficiency of Imperfect Shadow Maps for visibility calculation of Virtual Point Lights in our scenes. We are able to calculate diffuse and specular reflections for multiple indirect light bounces. Our approach works best in scenes containing geometry with a high diffuse reflection component and lower specular intensity, since surfaces with too high specular intensity can result in unintended sparkle artifacts.

Our implementation yet handles only one spot-light source, since this eases the way of performing impor-

tance sampling of VPLs over a limited area (in light view space) for the first bounce. In fact, directional light sources would work the same way using just orthogonal instead of perspective projection. Point lights on the other hand should be handled differently, since they would require eight Light-G-Buffers using the same G-Buffer setup. For this light source type, the use of two parabolic maps would be more encouraged, each one storing one of its hemispheres.

Extending to multiple lightsources, a linear relation between light source count and per-lightsource VPL density is expected, when maintaining equal frame-rates. However, some kind of importance sampling over the different light sources, which adjusts the number of VPLs seeded by a single light source depending on its total scene contribution (light source intensity, distance to the viewer), could be applied to speed up rendering.

All test scenes used in this paper, even those containing plane surfaces like the Cornell Box, are fully represented by point clouds. The implementation does not support polygon models for Global Illumination rendering yet. But since the integration of a polygon model support would require the sampling for surface points on this models anyway (ISM splatting), it would be more expedient to create a preprocessed point cloud representation of the model if possible.

References

[1] Stefan Brabec, Thomas Annen, and Hans peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *In Proc. of Computer Graphics International*, pages 397–408, 2002.

[2] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: Efficiently evaluating products of complex functions. In *Proceedings of ACM SIGGRAPH 2005*, 2005.

[3] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, 1986.

[4] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[5] Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 269–276, London, UK, 2001. Springer-Verlag.

[6] Martin Knecht. Real-time global illumination using temporal coherence. Master's thesis, Vienna University of Technology, 2009.

[7] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph.*, 27(5):1–8, 2008.

[8] Michael Wimmer and Claus Scheiblauer. Instant points, July 2006.

Interactive Ray Tracing of Distance Fields

Ondřej Jamriška*
Supervised by: Vlastimil Havran†

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

Abstract

Distance field is a flexible surface representation used in many applications. We study the algorithms for direct rendering of distance field surfaces representation via ray tracing in interactive frame rates. To accelerate the ray tracing we represent the distance fields by sparse block grid data structure. We compare the visual quality and the computation time for different methods of ray-surface intersection test and for surface normal estimation. We present a technique to accelerate the computation for primary rays from camera by projection. We also describe another technique for faster computation of shadow rays via volumetric occlusion. We show the results on four scenes of different complexities.

Keywords: Ray Tracing, Distance Field, Level Set

1 Introduction

Distance field is a versatile surface representation. Many applications need to represent dynamic surface that changes its shape over time in complex and unpredictable ways. For example, in solid modeling application, user may want to sculpt the shape of a surface by combination of adding material to existing model and carving holes into it. Another example is simulation of splashing water, where the interface between water and air needs to be tracked while it splits apart and merges together. In these circumstances, distance field representation is often used, due to its ability to handle operations that deform the surface and change its topology.

In addition to geometric modeling [2, 14] and simulation of liquids [8], applications utilizing distance fields include metamorphosis [3], geometric texturing [5], modeling of snow [10], and volume segmentation [16].

Image of a surface in distance field representation can be rendered using methods based on rasterization or ray tracing. Hardware acceleration makes rasterization an attractive option, however, due to the implicit nature of distance field representation, the surface must be first converted into set of rasterizable geometric primitives [12, 6].

*jamriond@fel.cvut.cz
†havran@fel.cvut.cz

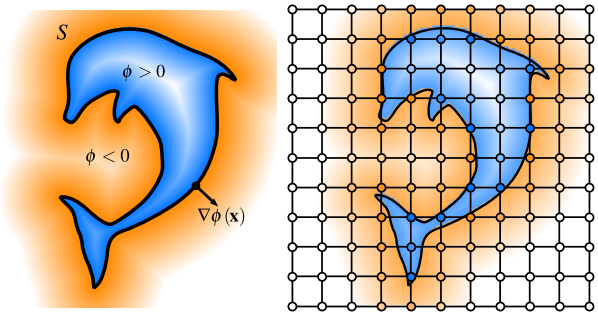


Figure 1: Distance field as a surface representation. Left: continuous distance field and a surface represented by its zero level set. Right: distance field sampled on grid.

Also, even though many sophisticated rendering techniques exist [9], the level of realism achievable with rasterization is limited.

Ray tracing is an image synthesis algorithm that is able to reproduce various optical phenomena. The algorithm traces individual paths of light as it propagates through scene. It relies on ability to compute the points where rays hit the surface. Since rays can be directly intersected with distance field, the costly conversion of surface into explicit representation is avoided.

In this paper, we will describe our raytracer that is capable of rendering surfaces in distance field representation at interactive frame rates. We designed it as a high quality rendering front-end for the kind of applications mentioned above. We thus assume that the surface is fully dynamic and its distance field can change each frame.

2 Distance Field

Distance field maps point in space to its shortest distance from nearest point on a surface. To measure the distance, Euclidean metric is often used. Distance field represents the surface in an implicit form, as a set of points that have zero distance to the surface. Example surface and its distance field is illustrated in Figure 1.

Surface S is defined as the zero level set of function ϕ,

$$S = \left\{ \mathbf{x} \in \mathbf{R}^3 \mid \phi(\mathbf{x}) = 0 \right\}.$$

Function $\phi : \mathbf{R}^3 \rightarrow \mathbf{R}$ is a signed distance function. At each point, $|\phi(\mathbf{x})|$ is the distance from $\mathbf{x}$ to nearest point $\mathbf{y}$ lying on the surface. By convention, the sign of $\phi(\mathbf{x})$ is negative when $\mathbf{x}$ is inside the surface, and positive when it is outside:

$$\phi(\mathbf{x}) = sign(\mathbf{x}) \cdot dist(\mathbf{x})$$

$$dist(\mathbf{x}) = \min_{\mathbf{y} \in S} ||\mathbf{x} - \mathbf{y}||$$

$$sign(\mathbf{x}) = \begin{cases} -1 & \text{if } \mathbf{x} \text{ is inside } S \\ +1 & \text{if } \mathbf{x} \text{ is outside } S, \end{cases}$$

where $||\cdot||$ is the Euclidean norm. Surface normal $\mathbf{n}$ corresponds to the gradient of ϕ,

$$\mathbf{n}(\mathbf{x}) = \nabla\phi(\mathbf{x}), \text{ and } ||\nabla\phi|| = 1.$$

For simple shapes, ϕ can be expressed in analytic form. However, for practical purposes ϕ is usually represented using set of samples located at discrete points in subregion of $\mathbf{R}^3$. Simple arrangement illustrated in Figure 1 puts samples on vertices of Cartesian grid. When value of ϕ is needed at point $\mathbf{x}$, it is reconstructed from nearby samples using interpolation. Trilinear interpolation calculates $\phi(\mathbf{x})$ as a linear combination of eight samples located at vertices of grid cell that contains $\mathbf{x}$.

3 Data Structure

Most applications need actual values of ϕ only at points that are close to surface, and just the sign of ϕ suffices away from surface. This allows for sparse sampling of ϕ, placing samples only inside narrow band around surface. Several data structures for sparse representation of distance field were proposed [4, 11, 13].

In our raytracer, we decided to use the sparse block grid data structure [4] for input distance field representation. Sparse block grid is easy to implement and allows random access in $O(1)$ time. Its memory consumption grows with $O(n^{2.25})$, where n is the number of grid cells along one axis [4]. Due to these properties, we assume it is likely that practical applications may utilize sparse block grid as their internal distance field representation.

Instead of storing samples on whole grid, sparse block grid divides the grid into coarse blocks and stores only samples on subgrids corresponding to blocks that are near the surface. The data structure is illustrated in Figure 2. Each block of coarse grid contains a flag and a pointer. The flag indicates whether the block is inside, outside or near the surface. If block is near the surface, its pointer points to a fine subgrid that stores the samples of ϕ.

4 Ray-Surface Intersection

Given ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with origin $\mathbf{o}$ and direction $\mathbf{d}$, we are looking for its first intersection with the surface on given interval $[t_{min}, t_{max}]$. Since surface is defined by the zero level set of signed distance function ϕ, intersections occur at roots of $\phi(\mathbf{r}(t))$. Thus, the task of finding the intersection is a root-finding problem. Specifically, for $t \in [t_{min}, t_{max}]$, we want to detect if at least one root of $\phi(\mathbf{r}(t))$ exists, and find the smallest t corresponding to the first root.

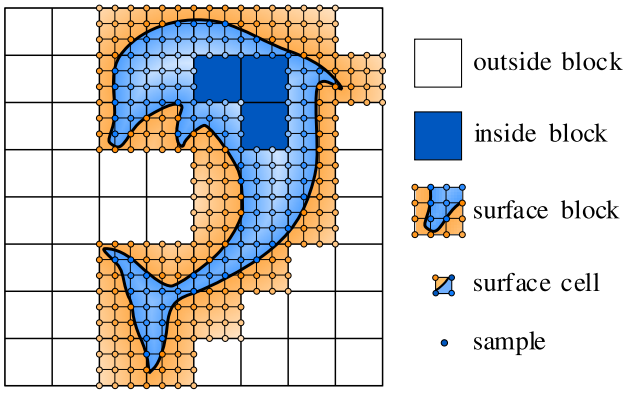


Figure 2: Compact representation of distance field using sparse block grid data structure.

4.1 Sparse Block Grid Traversal

The aim of traversal algorithm is to successively visit grid cells along a ray, identifying cells that contain the surface. For these cells, a test for an intersection needs to be performed. Traversal terminates as soon as the intersection is found or once the grid boundary is reached. The process is illustrated in Figure 3.

The traversal of sparse block grid consists of two nested loops. The outer loop iterates over blocks of coarse grid. When a visited block is entirely inside or outside the surface, the loop skips directly to the next block. Otherwise, the inner loop is invoked. Inner loop iterates over cells of a surface block and checks each cell if it contains the surface.

The check is based on comparison of signs of samples at cell vertices. Since the signed distance ϕ is negative inside the surface and positive outside, a cell contains the surface if the signs of samples differ. When such cell is visited, a test for intersection is performed. If ray intersects the surface inside the cell, the point of intersection is calculated.

In order to avoid fetching eight samples and checking their signs at each visited cell, we precompute a compact bit mask for every surface block. Each bit of the mask corresponds to one cell of the surface block. If cell contains the surface, its bit is set. Bit masks are recomputed every frame using single sequential pass over the cells of surface blocks. When the cell is visited during surface block traversal, we only test the corresponding bit in the block's bit mask to check if the cell contains the surface.

Grid traversal is initiated by intersecting ray with the grid's bounding box. We compute the points where ray

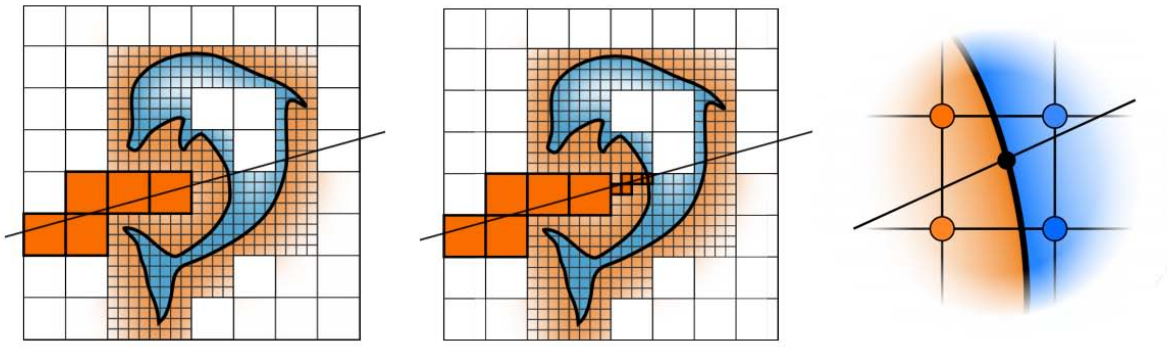


Figure 3: The process of finding the ray-surface intersection. The ray is first intersected with the grid's bounding box, and traverses blocks of the coarse grid (left). When the ray enters a surface block, it starts traversing the cells of the surface block's subgrid (middle). When a surface containing cell is encountered, the ray's intersection with the surface inside the cell is computed (right).

enters and exits the box using a ray–box intersection algorithm described in [17]. If ray has no intersection with the bounding box then it misses the grid and cannot intersect the surface. Otherwise, the grid traversal starts from the block that contains the point where ray enters the grid's bounding box. In case that rays origin is inside the grid, traversal starts from block containing the origin.

The algorithm for sparse block grid traversal is based on 3D–DDA algorithm [1]. At the beginning of sparse block grid traversal, we precompute several constants that can be reused during transitions between coarse and fine grid traversal in order to reduce their overhead.

4.2 Surface Cell Intersection

When ray enters a cell that contains the surface, its intersection with the surface inside the cell needs to be examined. The simplest method is to assume that the ray always hits the surface, and set the intersection point to the middle of points where ray enters and exits the cell. This method is fast, however, it produces artifacts that are visible especially on the contour of an object.

More accurate methods are based on finding the roots of $\phi(\mathbf{r}(t))$. Since ϕ is discretely sampled on grid, a continuous function needs to be reconstructed using interpolation. To reconstruct the value of ϕ at any point inside the cell, we use trilinear interpolation from eight samples at cell vertices.

Simple root-finding method first interpolates two values $\phi_{in} = \phi(\mathbf{r}(t_{in}))$ and $\phi_{out} = \phi(\mathbf{r}(t_{out}))$ at points where ray enters and exits the cell. If signs of ϕ_{in} and ϕ_{out} differ, the ray has hit the surface. The intersection point is calculated by approximating the position of a root as the point where line connecting values ϕ_{in} and ϕ_{out} crosses zero,

$$t_{hit} = t_{in} + (t_{out} - t_{in}) \frac{\phi_{in}}{\phi_{in} - \phi_{out}}.$$

Position of root can be further refined, as illustrated in Figure 4. First, the value of ϕ is interpolated at t_{hit}. Next, the interval $[t_1, t_2]$ is selected from two subintervals $[t_{in}, t_{hit}]$ and $[t_{hit}, t_{out}]$ such that $\phi(\mathbf{r}(t_1))$ and $\phi(\mathbf{r}(t_2))$ have different signs. A new root is then approximated as the zero crossing point of a line that connects values ϕ_{t_1} and ϕ_{t_2}. When repeated several times, this process is equivalent to a root finding technique called the false position method [15].

The convergence of false position method is only linear. This is not an issue, since we perform only small number of iterations anyway. We perform fixed number of 1-4 iterations because it is faster than using adaptive termination criterion, and visual results usually do not improve after more than 4 iterations.

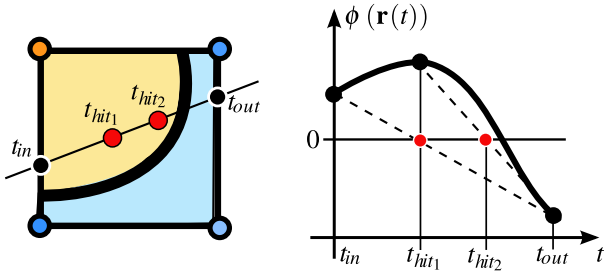


Figure 4: Locating the ray-surface intersection inside a cell. Left: ray passing through cell that contains the surface. Right: approximation of root position using first two steps of false position method.

5 Surface Normal Estimation

At the intersection point, a surface normal $\mathbf{n}$ needs to be evaluated for purposes of shading and spawning of sec-

ondary rays. In distance field representation, the surface normal corresponds to the gradient of the distance field. Since distance field is sampled on grid, the gradient has to be estimated using a combination of differentiation and interpolation.

We consider three gradient estimation schemes resulting from different ordering in which differentiation and interpolation is applied. The schemes vary in arithmetic complexity and in the number of samples they use. They also differ in the continuity of estimated normals, which affects the perceived smoothness of surface shape.

The first method is to find analytic derivatives of the interpolation filter. The filter is differentiated and analytical expressions of its partial derivatives are found. The components of $\nabla\phi$ are then obtained by convolving the samples of ϕ with filter's partial derivatives.

The second method is to first estimate partial derivatives of ϕ at sample locations using finite differencing. This yields values of $\nabla\phi$ at cell vertices. The $\nabla\phi(\mathbf{x})$ is then evaluated at $\mathbf{x}$ by interpolating gradients from cell vertices.

In the third method, the value of ϕ is first interpolated at number of points obtained by offsetting point $\mathbf{x}$ along coordinate axes. Next, $\nabla\phi(\mathbf{x})$ is computed by taking finite differences of interpolated values.

6 Acceleration of Primary Rays

Casting of primary rays can be accelerated by skipping the traversal of empty blocks that are outside the surface. This is achieved by reducing the $[t_{min}, t_{max}]$ interval so that the ray starts and stops close to the points where it enters the first surface block and leaves the last surface block.

To find reduced t_{min} and t_{max} for each primary ray, we rasterize surface blocks into min/max-buffers. Instead of rasterizing six faces of each surface block, we only rasterize a conservative bounding square of the block's projection onto the image plane. The center of a bounding square is computed by projecting the block's center using the camera matrix, and the width is determined based on the blocks space diagonal and the depth of its center. Each square has assigned value t that is equal to the distance from block's center to the camera.

Bounding square is computed for each surface block. Squares that correspond to blocks that are behind the camera are culled. Remaining squares are first sorted by their t values. Next, they are rasterized in back-to-front and front-to-back order into min-buffer and max-buffer respectively.

Since values t_{min} and t_{max} need not to be precise, the resolution of min/max-buffers can be lower than the image resolution. This can result in improved performance when there is large overdraw of bounding squares.

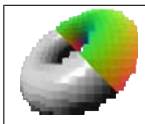
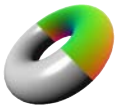
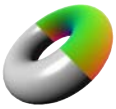
			
method	midpoint	1 iteration of false position	4 iterations of false position
#interpolations	2	2	5
time [ms]	8.0	8.4	11.2

Table 1: Surface cell intersection methods. Upper part of the torus shows normal in pseudo-color, lower part is shaded using Phong model.

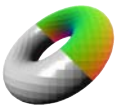
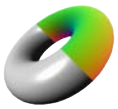
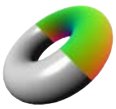
			
method	analytic derivative	difference of interpolations	interpolation of differences
#samples	8	16	24
#interpolations	0	3	1
time [ms]	7.1	9.5	8.4

Table 2: Surface normal estimation methods. Upper part of the torus shows normal in pseudo-color, lower part is shaded using Phong model.

7 Acceleration of Shadow Rays

Shadow ray query only has to detect if the ray intersects the surface, the actual point of intersection is not needed. This fact can be used together with the information already stored in sparse block grid to accelerate casting of shadow rays. Whenever ray passes through block whose flag indicates it is inside the surface, the ray must have also intersected the surface. We use a modified grid traversal to detect this situation and reduce the number surface cell intersection tests. This approach is inspired by the volumetric occluders technique described in [7].

The modified grid traversal first iterates only over blocks of the coarse grid. Traversal of a fine subgrid is not invoked when a surface block is visited. Instead, when a surface block is visited for the first time, the state of traversal variables is stored for a chance of later traversal restart. The traversal of coarse blocks then speculatively continues in the hope of encountering block that is inside the surface. The number of traversal steps elapsed from the first visited surface block is counted.

When the traversal visits a block that is inside the surface, the ray must have intersected the surface and traversal terminates. When the grid boundary was reached and no surface block was visited during traversal, the ray has no intersection with the surface and traversal also terminates. However, if a surface block was visited during the previous traversal and either the number of elapsed steps is greater than threshold n or the grid boundary is reached,

the normal grid traversal is restarted from the first visited surface block using the stored state of traversal variables.

8 Results

Our raytracer is implemented in C++. The raytracer is parallelized using OpenMP. We employ a simple parallelization strategy: we divide the whole image into square tiles of $N \times N$ pixels and assign the rendering of individual tiles to different threads.

Performance was measured on system with two Intel Xeon E5440 2.83 GHz quad-core CPUs with 6 MB shared L2 cache and 4×32 kB L1 data caches. In all tests the image resolution was 512×512 pixels, and tile size was set to $N = 32$ pixels. Rendering was parallelized using 4 threads.

The test scenes were prepared by converting triangle meshes into distance field representation at several different resolutions. We used models of Bunny, Dragon and Happy Buddha from the Stanford 3D Scanning Repository. The subgrid resolution of surface blocks was always set to 4^3 samples.

Three methods for surface cell intersection are compared in Table 1, with the number of trilinear interpolations they use for intersection evaluation inside single surface cell, and the total time to render the whole image. To compare the visual output we used a low resolution distance field of a torus. Even though the midpoint method is the fastest, it produces severe visual artifacts. These, however, tend to be less visible as the size of a surface cell projection approaches size of the pixel. Results of single and four steps of the false position method are visually indistinguishable. Thus, in all subsequent test we used single step of false position method for surface cell intersections.

Surface normal estimation methods are compared in Table 2. Normals calculated using the analytic derivatives of trilinear filter are discontinuous across surface cell boundaries, which is pronounced at specular highlights. The interpolation of differences is faster than the difference of interpolations even though it uses more samples. In subsequent tests we estimated surface normals using interpolation of differences.

The effect of primary ray acceleration using min/max-buffers is illustrated in Figure 6. The number of sparse block grid traversal steps is visualized using pseudo-color. The average number of traversal steps per pixel is clearly reduced with min/max-buffers.

We compared the rendering performance without primary ray acceleration and with the acceleration using min/max-buffers at three different resolutions. The results are summarized in Table 3. In this test, single primary ray is cast per pixel, and simple shading model is evaluated at the intersection point. The use of min/max-buffer led to improved performance in all cases.

In final test we evaluated the performance of shadow ray acceleration using modified traversal. For this test, shadow

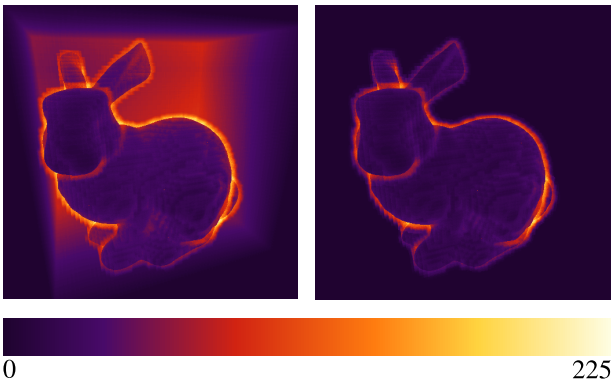


Figure 6: Comparison of the number of primary ray traversal steps per pixel with no acceleration (left), and with the acceleration using min/max-buffers (right).

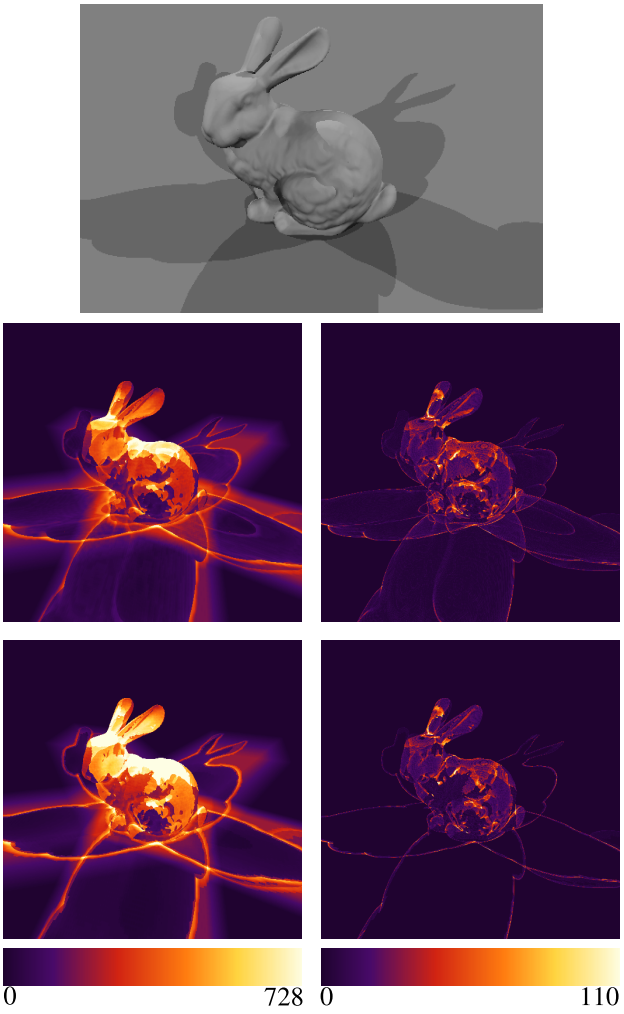


Figure 7: Comparison of the number of traversal steps and intersection tests for shadow rays using normal and modified sparse block grid traversal. Rendered image of the test scene is shown topmost. Top row: normal traversal. Bottom row: modified traversal. Left column: number of traversal steps per pixel. Right column: number of intersection tests per pixel.

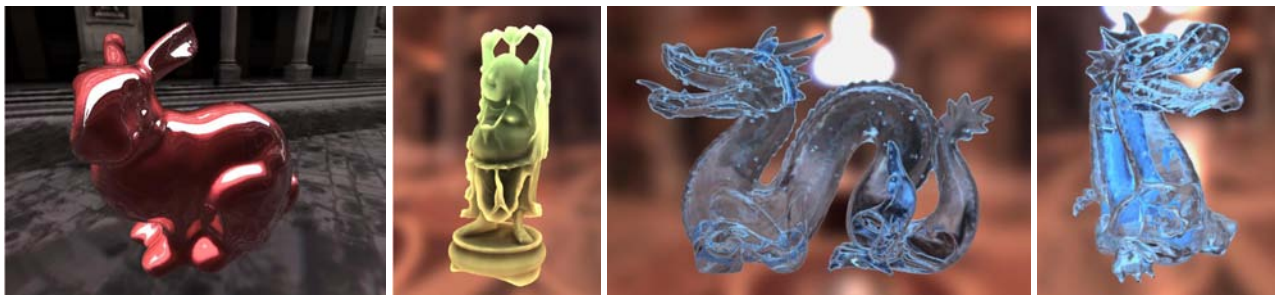


Figure 5: Three test scenes Bunny, Dragon, and Happy Buddha.

model	resolution	without min/max-buffers		256x256 min/max-buffers			128x128 min/max-buffers			64x64 min/max-buffers		
		#trav. steps [$\times 10^3$]	time [ms]	#trav. steps [$\times 10^3$]	time [ms]	speedup [%]	#trav. steps [$\times 10^3$]	time [ms]	speedup [%]	#trav. steps [$\times 10^3$]	time [ms]	speedup [%]
Bunny	$128 \times 128 \times 100$	5733	35.0	2885	28.6	18	3009	27.7	21	3247	28.3	19
Bunny	$256 \times 256 \times 200$	8915	45.0	2982	33.1	27	3186	32.3	28	3502	32.2	29
Bunny	$512 \times 508 \times 400$	15538	69.9	3056	51.1	27	3330	47.2	33	4287	50.0	28
Dragon	$128 \times 92 \times 60$	4264	28.8	2444	24.7	14	2538	23.9	17	2699	24.2	16
Dragon	$256 \times 184 \times 116$	6265	35.2	2536	28.1	20	2674	27.3	22	2902	27.4	22
Dragon	$512 \times 364 \times 232$	10435	50.4	2411	38.3	24	2620	35.7	29	3260	37.5	26
Buddha	$56 \times 128 \times 56$	3089	23.8	2225	22.1	7	2301	21.4	10	2409	21.6	9
Buddha	$108 \times 256 \times 108$	4146	28.0	2467	25.8	8	2568	25.9	8	2740	25.3	10
Buddha	$212 \times 512 \times 212$	6028	34.2	2360	33.9	1	2552	31.7	7	2973	32.5	5

Table 3: Comparison of primary ray casting performance without acceleration and with the acceleration using min/max-buffers.

rays are cast to five directional lights. Both the number of traversal steps and the number of surface cell intersection tests is recorded.

The effect of shadow ray acceleration is illustrated in Figure 7. The reduction of number of intersection tests is noticeable mainly inside large shadow areas on the ground plane. The number of traversal steps is locally lowered at some places and raised at other.

We compared the performance of the shadow ray casting with and without shadow ray acceleration. The results are summarized in Table 4. Tests were performed for two values of threshold n used in modified traversal. The threshold determines for how many steps the coarse grid traversal continues after the first surface cell is visited until it is restarted.

Although the number of intersection tests was always reduced using the modified traversal, the number of traversal steps increased in some cases due to traversal restart, leading to worse performance than that achieved with normal traversal.

9 Conclusions and Future Work

In this paper we have presented the techniques behind our interactive distance field raytracer. Basic methods for ray-surface intersection and surface normal estimation were described and compared. We have shown two simple acceleration techniques for casting of primary and shadow rays and analyzed their performance on three test scenes. In all tests we achieved reasonable frame rates ranging from $10 \sim 50$ fps.

In future work we would like to investigate means of accelerating the search for intersection either by augmenting the coarse grid with additional information encoding the empty space, or by building a hierarchical spatial subdivision data structure on top of the sparse block grid.

Acknowledgement

This work has been partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS 13139/10/800890 and by the Ministry of Education, Youth and Sports of the Czech Republic under the research program MSM 6840770014 and LC-06008 (Center for Computer Graphics), the Aktion Kontakt OE/CZ grant no. MEB 060906.

model	resolution	no acceleration			n=12				n=32			
		#trav. steps [$\times 10^3$]	#isect. tests [$\times 10^3$]	time [ms]	#trav. steps [$\times 10^3$]	#isect. tests [$\times 10^3$]	time [ms]	speedup [%]	#trav. steps [$\times 10^3$]	#isect. tests [$\times 10^3$]	time [ms]	speedup [%]
Bunny	$128 \times 128 \times 100$	7639	529	66.4	8259	348	59.4	10.5	8876	320	59.7	10.1
Bunny	$256 \times 256 \times 200$	11153	542	82.8	11663	345	72.8	12.2	12360	294	71.6	13.6
Bunny	$512 \times 508 \times 400$	18586	472	123.4	19137	292	108.9	11.7	20106	232	106.6	13.6
Dragon	$128 \times 92 \times 60$	6694	426	56.6	8145	365	56.7	-0.2	8548	342	56.7	-0.2
Dragon	$256 \times 184 \times 116$	9708	433	68.6	10791	333	64.7	5.7	11731	285	64.5	6.0
Dragon	$512 \times 364 \times 232$	15881	377	96.5	16693	264	87.0	9.9	17550	197	84.4	12.5
Buddha	$56 \times 128 \times 56$	734	104	16.6	1015	100	17.9	-7.9	1030	100	17.9	-8.3
Buddha	$108 \times 256 \times 108$	1110	116	19.7	1471	107	21.2	-7.6	1620	106	21.8	-10.4
Buddha	$212 \times 512 \times 212$	1581	96	24.6	1963	84	26.0	-5.9	2295	78	26.6	-8.2

Table 4: Comparison of shadow ray casting performance using normal and modified traversal.

References

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Proc. Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, August 1987.

[2] J.A. Bærentzen and N.J. Christensen. Volume sculpting using the level-set method. In *International Conference on Shape Modeling and Applications*, pages 175–182, 2002.

[3] D.E. Breen and R.T. Whitaker. A Level-Set Approach for the Metamorphosis of Solid Models. *IEEE TVCG*, 7(2):173–192, 2001.

[4] R. Bridson. *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, 2003.

[5] A. Brodersen, K. Museth, S. Porumbescu, and B. Budge. Geometric texturing using level sets. *IEEE TVCG*, 14(2):277–288, 2008.

[6] C.S. Co, B. Hamann, and K.I. Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. In *Proceedings of the 11th Pacific Conference on Computer Graphics*, pages 325–334, 2003.

[7] P. Djeu, S. Keely, and W. Hunt. Accelerating Shadow Rays Using Volumetric Occluders and Modified kd-Tree Traversal. In *Proceedings of Conference on High-Performance Graphics 2009*, pages 69–76, 2009.

[8] D. Enright, S. Marschner, and R. Fedkiw. Animation and Rendering of Complex Water Surfaces. *ACM Transactions on Graphics*, 21(3):736–744, 2002.

[9] R. Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[10] T. Hinks and K. Museth. Wind-driven snow buildup using a level set approach. In *Eurographics Ireland Workshop Series*, pages 19–26, Dublin, Ireland, 2009.

[11] B. Houston, M.B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics*, 25(1):151–175, 2006.

[12] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM.

[13] M.B. Nielsen and K. Museth. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3):261–299, 2006.

[14] R.N. Perry and S.F. Frisken. Kizamu: A system for sculpting digital characters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 56. ACM, 2001.

[15] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[16] R.T. Whitaker, D.E. Breen, K. Museth, and N. Soni. A framework for level set segmentation of volume datasets. In *Proceedings of ACM International Workshop on Volume Graphics*, pages 159–68, 2001.

[17] A. Williams, S. Barrus, R.K. Morley, and P. Shirley. An Efficient and Robust Ray–Box Intersection Algorithm. *Journal of Graphics Tools*, 10(1):49–54, 2005.

Applications

Concept of Interactive Coloring book

Tomáš Pastorek*
Supervised by Petr Felkel†

Faculty of Electrical Engineering
Czech Technical University in Prague
Czech Republic

Abstract

Coloring book programs are often simple and do not develop anything but mouse-move-and-click skills. The only task is to fill-in predefined regions by a selected color. This research exploits the possibilities offered by a computer to do more.

Based on discussions with a clinical psychologist, we designed an interactive coloring book concept to help preschool children in development of creative and logical thinking and basic computer operation skills. Children are offered a simple scene for coloring where individual objects can interact with their mouse input. The objects behave exactly as the children know from their every day life. The object reaction is physically simulated.

We present a pilot implementation of the interactive coloring book concept. The scene can be designed using elementary objects, images, sounds, and logic, allowing more complex entertainment. Scene description is stored in XML files, being simply editable and manageable.

We have successfully validated our concept by a user study involving 5 preschools and their parents. The good experiences lead to continue the coloring book development in cooperation with Faculty of Education, Charles University in Prague.

Keywords: Coloring book, Physics, Physical Simulations, Interactivity, Children, Games

1 Introduction

Coloring books are extremely popular among children, as most every child loves coloring. This simple and fun activity has also a positive effect on child development - it improves their graphomotor skills – their hand-eye coordination. These skills are later used for writing, drawing and other manual tasks.

Children of age 4–5 like tangible images, preferably with an already filled example, and basic colors. Undo function or an eraser can confuse them. Preschools (5-6 years) can handle more complex images, more color tones, and should be able to orient themselves in space (up, down, left, right). They have developed an abstract thinking and do not need a colored example. Preschools can solve simple puzzles and quizzes.

Software implementations of coloring books as they are now available as web pages are only an imitation of paper coloring where the mouse clicks simulate surface coloring with a pencil. A complex movement with a pencil is replaced by simple clicking and the graphomotor learning effect of precise coloring is nearly lost.

The computerized colorings should bring something more, but the colorings available in the internet rarely bring anything really new in comparison with the paper ones. Based on discussion with a children psychologist, we proposed a concept of improved interactive coloring book including animation, physical simulation of moving objects, and programmable scene logic.

Introducing animation and physical response to the current coloring books brings an interesting combination that can attract children and certainly can develop more skills—from fantasy to logical thinking [1]. In addition, soundtrack gives an additional element to which children must respond. It is not a mere filling up of closed regions we know from nowadays coloring books, but the overall concept of a simply extendable multimedia software solution for the children entertainment.

2 Review of similar projects

Present coloring book applications are seldom downloadable as standalone programs. It is common now to present them in the form of web applications. The criteria used for evaluation of coloring book programs included the following ones:

Interactivity - should give more than just coloring by clicking.

Controllability – The user interface and usability for coloring must be in relation to the children age, for young kids as simple as possible without text. It should be understandable by the parents who help with the rules.

Image for coloring – should not contain small areas hard to target by the mouse cursor. Older children can have

*pastoto1@fel.cvut.cz
†felkepet@fel.cvut.cz

more complex images. The child should have a possibility to choose image for coloring from the set of images.

The color palette – should also respect the children age and their ability to designate the colors. Small children need basic colors and large color spots in the palette, older children can use pastel colors and small color spots.

Graphic design – should be eye-catching and attractive.

Programming grade - applied technology and amount of bugs in the application.

We have tested 15 coloring programs, five of them in greater detail (Interactive coloring book from Keith Haring Foundation [2], Coloring books at Apples4theteacher.com [3], National Association of Home Builders – coloring book [4], Fisher-price Click and Color [6], and Coloring book at Alik.cz [7]). Rating according to the criteria is in Figure 2 For additional details, see [16].

Interactive coloring book from Keith Haring Foundation [2] is an artistic group project. This group is organizes charitative campains focused on AIDS and children. Unfortunately, the main attention is fixed on the artistic aspect leaving out the usability and amusement for small children. Unfortunately, the control is too complex also for the school children.

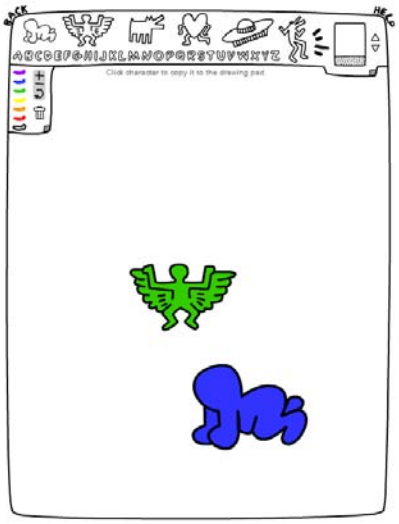


Figure 2: Interactive coloring book from Keith Haring Foundation

Coloring books at Apples4theteacher.com [3] is a very interesting project, outstanding in the number of images offered for coloring. A large number of colors in the palette could be a dissadvantage for small children, as they cannot name all of the colors. On the other hand, the artworks are simple with large regions, which is good for small children.

Coloring book from National Association of Home Builders [3] is a complement service of NAHB oriented on construction, providing a small set of artwork for coloring. The way of coloring is distinct from others. It is a drawing by mouse with a trespasing control, avoiding getting across the region outline. The game offers a group of tools, including eraser and the brush stroke size. These features together with a large amount of colors fit to older children. The different way of coloring is more entertaining.

Figure 3: National Association of Home Builders coloring book

Fisher-price Click and Color [6] serves as part of web presentation of the company producing toys [5]. They offer online games for children of different age groups. Their colorings have very cute colors and real-looking palette and brush-shaped cursor. These features nicely adapt the user interface to the preschool children needs.

Coloring book at Alik.cz [7] is a Czech project of the Mafra company. It suffers from several problems, like wrong cursor shape (cross instead of pencil or brush) unsuitable for small children and an error in the code—it is impossible to recollor the region filled with a black color.

2.1 Conlusions from the review

Preschool children prefer large regions and images of real things, they know from everyday life. Such themes help them in selection of appropriate colors. Colored example images also serve well for inspiration. Palette should contain basic colors only, those the children can call by name. More complex tools such as the *rubber* or *undo* confuse the children and should be avoided.

Primary school children can cope with more complex images; physical simulations can contain simple mechanisms and logical quizzes. Palette can contain much more colors. Colored example images are not necessary. School children have enough fantasy and can improve their sense for art.

The pilot program should contain predefined palettes for different ages. The spots should be large to simplify the selection of colors. School children should get a color name

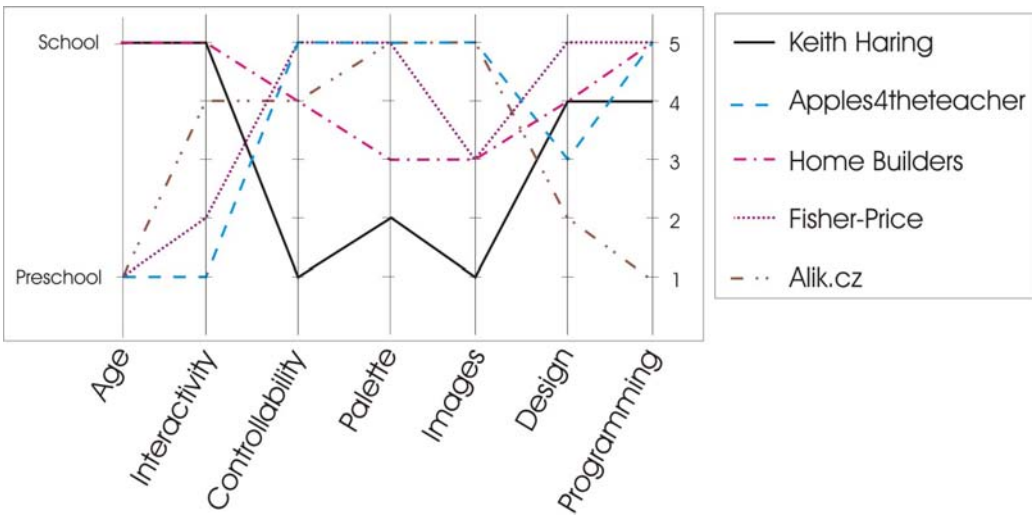


Figure 1: Rating of five coloring programs based on different criteria (1-the worst, 5-the best)

hint when the cursor stops on the color spot. The cursor must be large and recognizable to help selecting the colored regions. The environment should be cute and intuitive. The style of the image surrounding should be minimalistic, to avoid disturbances. A set of predefined scenes for different ages should be prepared.

3 Method

Based on the review of existing coloring applications we have designed a concept of interactive coloring book. The solution must allow a high level of configurability to match the needs for a larger range of children ages - from preschools to school children. The program should not limit itself only to coloring. It should contain tools for simple programming, applicable for a simple game development. Logical quizzes and different methods for coloring should be easy to implement. To fulfill these requirements, we have created a controller.

3.1 Controller

Controller is a dynamic library (assembly) loaded together with the scene. It containes the control logic of the scene. Controler can create objects, set their specific attributes, play sounds, etc. Each object is addressed by its name, defined in XML in the scene definition file. Coloring program creates new instances of objects by reflection.

Comunication between the sceene and the objects is accomplished by event handling. For this purpose, the controler defines a fixed communication interface `IControler`. Overview of the whole application architecture is in Figure 4.

The role of the controler can be demonstrated on the following an example situation: The scene contains a small disc, which should turn to red color after the mouse click. Relevant implementation of the controler would look as follows. The handler of the *Insertion of the object into the scene event* is created. While creating the `disc` object, this event handler connects the *mouse click event* with it. Body of the *mouse click event* handler would contain the color-change command of the `disc`.

We can chose from the following events while defining the scene:

- `Scene` events
  - *SceneLoaded* - scene loading is finished
  - *ObjectLoaded* - object loading is finished
  - *ObjectAdded* – object has been added to the scene collection
  - *ObjectLeftScene* - object left the visible scene (useful for object recycling)
  - *ObjectCollided* - object collides with another object
- `Object` events - can be registered for each individual object
  - *MouseDown* - mouse click on the object
  - *MouseUp* - mouse release on the object
  - *MouseMove* - mouse move on the object
  - *BackgroundChanged* - object was assigned a new background color
- `Palette` event
  - *SelectedColor* - notification of the color selection

The system of scene control as described here is very important for the universality of the coloring application. The scene logic is not fixed, but it can be different for each scene.

3.2 Scene definition

Visual appearance of the scene is defined in XML document. XML was chosen as a simply human editable format as the scene editor was not part of this phase of the project. XPath technology [17] was selected for reading of XML files. It allows for fast access to particular elements in the document.

Every object in the scene has the properties defined in the XML file: Visual properties (such as width, height, position, rotation, color or image fill), physical properties (weight, vector of weight distribution), and other properties (data type including its namespace, object name for object identification in Controller).

The object definition contains only the name of data type (class name with namespace). This allows for introduction of new data types without a necessity to change the XML document scheme. Based on the data type name, new instance of object is created by reflection while reading the scene description. Our concept of interactive coloring book contains the following predefined objects:

- `ColoringBook.Objects.Rectangle`
- `ColoringBook.Objects.Ellipse`
- `ColoringBook.Objects.Polygon`
- `ColoringBook.Objects.Vehicle` – simple physical model of a truck
- `ColoringBook.Objects.Tire` – car wheel

4 Program architecture

The display layer (implemented by the Windows Presentation Foundation technology - WPF), receives the list of objects (interface `IObjectBase`) and the list of colors in the palette (`PaletteColor` class) via the `ObservableCollection`. It is a specialized collection able to announce the changes for the display. The changes are signaled to the display layer by means of *PropertyChanged* event.

Newton Dynamics physical engine is connected with the rest of the coloring application by the Proxy design pattern (classes `CWorld` – the physical world – and `CBody` – representing the object instance in the physical engine). Scene object for logic processing uses only the known interface `IController` – thanks to that, it is possible to create different scene logics by means of different implementation of this interface. See Figure 4.

5 Results

We present a pilot implementation of the interactive coloring book concept for Windows platform. This operating system has been selected based on the world statistics of operating system usage by the children target group.

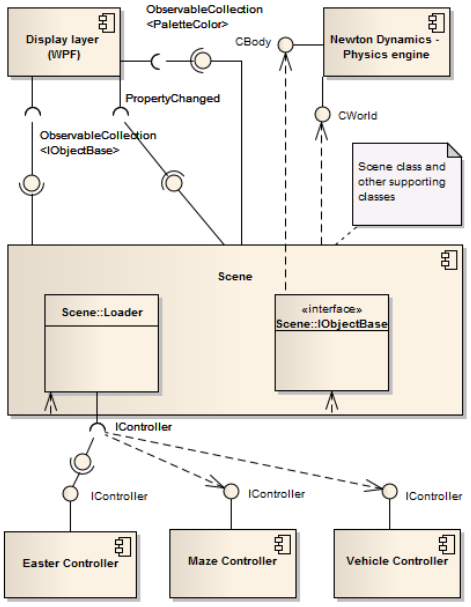


Figure 4: Component diagram depicting of interactive coloring book architecture. `IControler` describes the logic of each scene

Development was done in C# language under the .NET Framework 3.0.

The Windows Presentation Foundation (WPF) was selected for rendering. It allows for direct rendering of vector graphics and for separation of application and presentation layers. By means of XML markup langue XAML, we can define the application appearance independently of the code, the styles for elements dynamically, and create relations to database sources. This working style is similar to the working style of web designers. Another advantage is a simple portability to Silverlight technology.

As a physical engine, we have selected the Newton Dynamics [14] with a .NET wrapper [15]. This Physical engine is stable, fast, and well documented. With the .NET wrapper the engine is usable in the selected technologies (.NET, C#, ...) and provides a good set of sorts of joins. Integration of WPF with the Newton Physical Engine is very simple. The physical engine returns the positions and rotations of objects in matrix form. These matrices can be assigned as transformation matrices to the objects in WPF. We just have to use the same scales for both (see [13] for details).

5.1 Implemented scenes

To demonstrate the abilities of our interactive coloring book concept we created four example scenes. An Easter Eggs scene, Path Coloring, Truck, and a Music scene.

Easter eggs scene The task of this scene is to fill in the eggs ovals with an appropriate color. The steps the eggs lay on can be colored anyhow. After filling in,

the eggs start rolling and falling down to the bunny. The moment all the eggs fall down to the bunny the fanfare is played and the game is over. The eggs scene with newly assigned colors appears and a new game startes. The video of this scene can be seen on [19], the scene preview is in Figure 5.

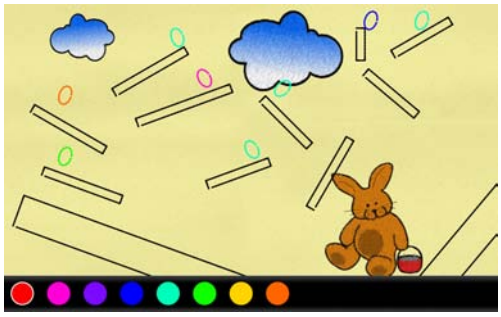


Figure 5: Easter eggs scene

Path Coloring In this scene, the user must create the path for the marble to get it into the house. The path is built by coloring the rectangular steps. The marble can roll on the filled steps only. Non-filled steps does not support the ball and it falls down. The correct filling order is: all the steps first, then the marble to release it onto the path. Fall out of the scene and reaching the house play a sound and the game starts again. The video of this scene can be seen on [20], the scene preview is in Figure 6.

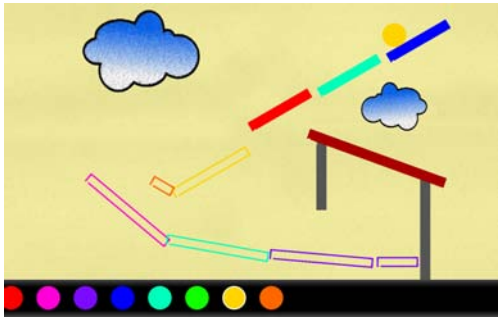


Figure 6: Path scene

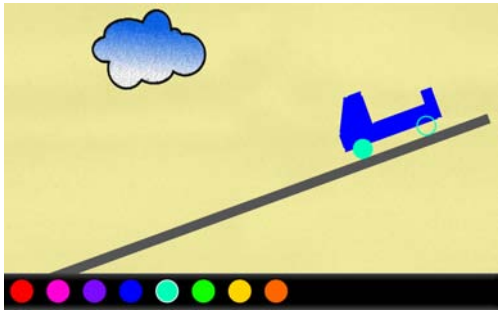


Figure 7: Truck scene

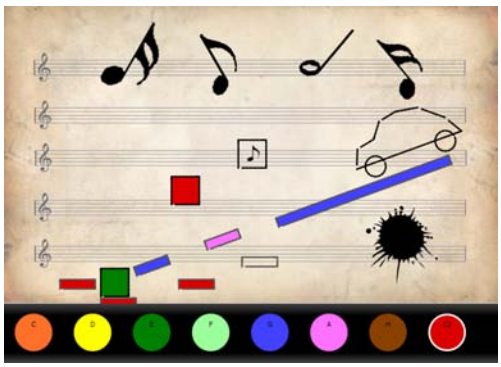


Figure 8: Music scene

Truck Scene This simple scene shows how to create and control vehicles. After coloring the truck body and wheels, the motor turns on and the truck drives down the ramp. When it reaches the scene border, colors are reset and game starts again. The video of this scene can be seen on [21], the scene preview is in Figure 7.

Music scene The task is again to create a path for a vehicle. After coloring the path pieces and after filling the holes by falen collored rectangles, the car can be filled in and it starts moving down the ramp. Each ramp piece sounds different tone, given by the applied color. If any noncolored path piece remains, the car cannot start and honks. The video of this scene can be seen on [22], the scene preview is in Figure 8.

5.2 Mini usability study

Usability of the product can be defined as the grade people can use the product to fulfill given tasks [18]. We have performed a very limited low cost low effort mini usability study. By means of this mini usability study we wanted to find out, how the users liked the user interface, how they reacted on different scenes and what experience the scenes left behind in their minds.

Interactive coloring book program was tested by five users: two girls, 5 and 7 years old, with previous experience with a computer; two boys, 4 and 7 years old, with no and very little experience with computer; one woman, 39 years old, with computer skills. The user study was performed on the first three scenes.

We have used a full screen mode in the user study, enlarging the regions for coloring and avoiding erroneous clicking outside the coloring book program. The evaluation included: a) Work without any help, where the user should find out, what is the goal and how to reach it, b) work after possible oral help, c) selection of appropriate colors, d) coloring itself, and e) final oral evaluation.

Main problems arising from the user study included:

- At the beginning, it is not very clear what should be

done. Sometimes, oral help is necessary for understanding the goal, especially for the younger children.

- The pencil (cursor) does not change its color. Children have problems to recognize which color is selected.
- By color selection, they did not recognize, that the point of the pencil (cursor) is the selection tool.
- Marking of the selected color in the palette by a white border is not sufficient, as the children took time to recognize, that the selected color is marked by a white circle.
- Some color tones are very similar and therefore hard to distinguish.

The children liked the scene with a truck the most because its coloring is simple and the finished task is rewarded by an engine start followed by the truck movement.

6 Discussion

The potential of the interactive coloring book concept can be used for development of simple entertaining games for healthy children. It can be employed for learning basic computer skills through play, for logical thinking improvement, and for phantasy development.

In cooperation with experts on special education, we can create puzzles focused on children with light dissability. Nowadays, we work on extension of interactive coloring book concept for visually impaired children in joint project with Faculty of Education. There is lack of such oriented software.

Simultaneously, we do not limit us to coloring books. It is possible to create simple games exploiting the physical engine.

It proved in practice, that the selected technologies were chosen well and they exactly matched our assumptions. Only if we decide to port the Interactive coloring book to web platform Silverlight, we would have to change the physical engine (e.g., to Farseer Physics [23]).

As the thorough reviewer correctly mentioned, "there is a real danger that children will lose the ability to hand write if they are constantly just using a mouse and keyboard". A different method for coloring, e.g. such as stylus or computer with a touch screen like TabletPC should be designed and tested. Having no such specializad hardware we have not addressed this in our study and implementation, leaving it for possible future work.

Usability tests did not reveal any serious problems related to the concept or implementation of this project. All issues that appear can be solved by scene file modification or by small changes in the program. Usability tests verified that our introductory presumptions were correct. Children accepted physical simulations well knowing such behavior from everyday experience in the real world. Simple and intuitive user interface is also important, mainly for the first steps with the new program. Even though the mini user study showed minor imperfections, the children accepted the interactive coloring book very well.

7 Conclusions

We have presented a concept of interactive coloring book and a pilot application based on this concept. They were designed in cooperation with a clinical psychologist with the main idea in mind – how to use computers in children entertainment better. The classical paper-and-pencil coloring scheme cultivating imagination and sense for colors was enriched by the physics simulations helping the children to understand the interactions among objects in the real life.

The capabilities of the interactive coloring book were demonstrated on four scenes. The Easter egg scene prepared for the smallest children was a simple coloring by appropriate color. Next scene contained a ball and the path split into parts. The parts had to be colored before releasing the ball to get it to the end post. The last two scenes contained a ramp and a vehicle. The truck in the former scene started moving down the ramp after filling up the ramp and the truck body. The latter scene with a car was enriched by playing tones according to the color selected for coloring.

In principle, it is not complicated to create a new scene with any functionality, because the scene logic is stored in an XML file and not fixed in the program. The pilot application tested the concept of interactive coloring book and proved its popularity among the children.

8 Future work

After the validation of interactive coloring book concept, we concentrate on creation of the development environment for scene creation and scripting. Such a tool will serve for fast creation of simple games with physics simulation. Simultaneously, in cooperation with the special education department at the Faculty of Education, Charles University in Prague, we work on the set of games and research tools for visually impaired children.

Acknowledgements

The authors wish to thank the psychologist Naďa Kravincová for professional consultancy of our ideas and focusing on the children needs, and our usability testers Clara, Jonas, Mathew, Betty, and Alice. This research has been partially supported by the MSMT under the research program MSM 6840770014.

References

[1] Naďa Kravincová. Psychologist. Pesonal discussions and consultations. 2009

[2] *Keith Haring Interactive Coloring Book.* http://www.haringkids.com/coloringbook/index.html

[3] *Apple4teacher.com* http://www.apples4theteacher.com/coloring-pages/

[4] *National Association of Home Builders: Coloring Book.* http://www.nahb.org/coloringbook/book.aspx

[5] *Fisher-Price. Online Game & Activities,* 2008, http://www.fisher-price.com/us/playtime/

[6] *Online Coloring,* http://www.fisher-price.com/us/littlepeople/clubhouse/games.asp?section=onlinecolor&gameID=LP_OnlineColoring

[7] *Colorings on Alík.cz,* 2008, http://www.alik.cz

[8] *Crayon Physics,* Kloonigames, 2008, www.crayonphysics.com, www.kloonigames.com/

[9] *Microsoft Physics Illustrator for Tablet PC, 2004,* http://www.microsoft.com/windowsxp/downloads/powertoys/tabletpc.mspx#ETBAC

[10] *Software Logopedy,* multimedia ART, In Czech. http://www.jablko.cz/.

[11] *Software Reader, help withUFFF dyslexy,* multimedia ART, In Czech. http://www.jablko.cz/Citanka/default.htm

[12] *Software Mathematics 1–5, help with dyscalculia,* multimedia ART, In Czech. http://www.jablko.cz/matematika/default.htm

[13] *Chris Cavanagh blog* http://chriscavanagh.wordpress.com/

[14] *Newton Dynamics Physical Engine.* www.newtondynamics.com

[15] *.NET wrapper* http://pagesperso-orange.fr/flylio/Programmation/NewtonWrapper.zip.

[16] Pastorek, T. *Interactive coloring book.* Bachelor thesis. CTU FEL, 2007, In Czech.

[17] Clark, J., SeRose, S. *XML Path Language (XPath).* Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath/

[18] Maly, I. *Usability of (web) applications,* 2006, In Czech. http://webing.felk.cvut.cz/old/output/pub/usability_web_app.pdf

[19] *Easter eggs scene video.* http://www.youtube.com/watch?v=NBhzxPU1mvU

[20] *Path coloring video.* http://www.youtube.com/watch?v=OyoRTU0xf-g

[21] *Truck scene video.* http://www.youtube.com/watch?v=oXRVD6Ka44I

[22] *Music scene video.* http://www.youtube.com/watch?v=MfEc0HJBYww

[23] *Farseer Physics Engine 3.0* http://www.codeplex.com/FarseerPhysics

Obesity in Children - A Serious Game

Elmedin Selmanovic*
Supervised by: Kurt Debattista,† Simon Scarle,‡ Alan Chalmers§

International Digital Laboratory
University of Warwick
Coventry / United Kingdom

Abstract

Childhood obesity is a prevalent problem in most developed countries. It can have a significant negative impact on a childs health including diabetes and cancer. Help in the preventing and reducing obesity is required. One possible method suggested in this paper is a serious game, which would increase energy expenditure during play, educate about nutrition and promote healthy eating and physical activity. Methods and ideas, informed by the related research, which were used for the implementation of such a game, are presented in this paper. Although not a complete solution in itself the game could help in the fight against childhood obesity.

Keywords: Serious Games, Obesity, Edutainment, Exergaming

1 Introduction

The prevalence of childhood obesity in advanced countries is an increasing problem. The number of obese children varies from 17% in the UK [2] and 16% in the US [21] to 12% in Australia [35]. Obesity can have negative effects on a child's health (e.g. diabetes, cancer and cardiovascular disease) and negative psychosocial impact including low self esteem and stigma [6, 7]. Moreover, in 40% to 70% of the cases, obesity is likely to persist into adulthood [24] with accompanying health risks and possible socioeconomic problems.

The causes of childhood obesity are complex and multifactorial including unhealthy eating patterns and an inactive lifestyle, both of which can be linked to increase in the time spent watching television and playing video games [6, 23]. Given that an average US family keeps its TV turned on 8 hours a day, the time a child spends in front of the screen can reach up to 55 hours per week [37]. One response was reduce the amount of TV watching, but different solutions were needed, as children did not relinquish screen time that easily [8, 11]. One possible answer to this problem would be to convert this sedentary screen time into a more active form and use it to promote physical activity and healthy eating.

The aforementioned objective could be accomplished using a game. This would fall into the domain of serious games, games whose primary function is other than pure entertainment. The aim of this paper is to show the creation process of the game that can help tackle and prevent childhood obesity.

First, definition and brief history of serious games is discussed. Related work is provided in Section 3. In Section 4, the game design and different considerations are explained. Section 5 deals with the actual implementation and examines a selection of various tools and techniques. Feedback obtained from the first game testing is provided in Section 6. The conclusion is presented in Section 7 and future work is discussed in Section 8.

2 Serious Games Background

Clark Abt published a book entitled Serious Games in 1970 (Abt 1987). This was the first time the term was used. Although, in this book, Abt was mostly concerned with card and board games, the definition which he offered is still relevant: *"Games may be played seriously or casually. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement. This does not mean that serious games are not, or should not be, entertaining."*

However, today, serious games do not have such a clear scope, because the majority of researchers involved tend to offer their own, somewhat, different definition. For instance, not everyone agrees that factor of fan or entertainment has to be included. While Zyda [39], Bergeron [1] and Prensky [22] think it is necessary, Stoll [31] completely rejects the idea of putting fun into the learning process. Michael and Chen [19] take a more moderate view suggesting that fun although desirable is not essential. According to Zyda [39] an additional factor that has to be considered is pedagogy, but this still needs to be subordinate to entertainment. Bergeron [1] finds it crucial that a scoring system and challenging goals are included. The

*elmedin.selmanovic@warwick.ac.uk
†k.debattista@warwick.ac.uk
‡s.scarle@warwick.ac.uk
§a.g.chalmers@warwick.ac.uk

Serious Games Initiative [14] focuses on the usage of serious games for education, training, health and public policy. Still one common idea can be taken out and used as the broad definition of serious games; those are the games whose main purpose is other than pure entertainment.

With such a broad definition the history of serious games is difficult to pin down. However, factors that affected the development of this field can be identified and include the evolution of industrial-military complex, a usage of digital technologies in medical education, expansion of the computer industry and the popularity of commercial games [1]. The first instance of serious gaming can be traced back to Edwin Link who constructed the first flight simulator in the late 1920s [16]. After this early start sporadic examples of serious games followed mostly in the military and medical fields. In the 1980s two promising disciplines of virtual reality and edutainment, which used games for non-entertainment purposes, started to expand [32]. Unfortunately, this expansion did not last long and both of these concepts failed to fulfil their early promise. Virtual reality struggled with expensive hardware, the absence of case studies, lack of attention for end user's requirements and failures to provide meaningful and functional intellectual property [32]. At the same time edutainment, the idea of education through entertainment, produced boring games no one had interest in playing [36]. Serious games have to avoid these pitfalls to succeed, especially given that, today, edutainment is considered its key part. The expansion of serious games started in 2002 with lunch of the Serious Games Initiative which tries to form connections between game industry and health, education, training and public policy projects that require its services. Serious games have a large application area including military, government, corporations, healthcare and cultural heritage. The market size of serious games is not easily determined (one of the reasons being the broad definition), but the U.S. military alone spends millions on their development annually [1].

The advantages that serious games have to offer need to be defined. What is clear is that a game's simulated environments provide the user with the opportunity to experience situations which cannot be replicated in real world due to time, cost and safety reasons [4, 30]. However, the effect that they have on players is difficult to assess because complexity of the games requires variables that are narrowly defined and conditions that are tightly controlled, resulting in research with rather limited claims [36]. Nevertheless, it has been shown that serious games can improve analytical, spatial and strategic skills, learning and recollection capabilities, psychomotor skills, visual selective attention, self-monitoring, problem recognition and problem solving, decision making, short and long term memory, social skills such as collaboration, negotiation and shared decision making [20, 25]. It is important, in this context, to recognize that not every serious game offers all of these benefits and each one should be assessed individually to determine success.

3 Related Work

There are a number of studies that support the idea of exergaming: using video games for exercises [13, 17, 27, 29]. Most of them test the amount of energy spent during play and compare the result to both sedentary screen time as well as sports activities. The active games significantly improve energy expenditure, sometimes even doubling it compared to the regular sedentary screen time, while in most cases they are still not as good as doing regular sports. A comparison which demonstrate how exergaming can come close to the real sports, shows that playing Wii Sports Tennis spends 750 kJ/hour while real bowling spends 800 kJ/h [12].

In most cases of exergaming, specialist hardware is required. The idea of connecting exercise equipment to video games is almost as old as games themselves [28]. An early commercial example is the project named Atari Puffer where an exercise bike was interfaced with a game console. After this, a series of companies started using exercise bikes for the same purpose with today's representatives: Tacx Fortius Trainer, Fisher Price Smart Cycle and Cateye GameBike Pro. Foot operated pads are another type of devices used for exergaming. One of the especially successful games that used foot operated pad was Konami's Dance Dance Revolution with 6.5 million copies sold. Successful instance of this device is Nintendo's Wii Balance Board [28]. Lastly, we have equipment that uses motion sensors for input; popular examples are: Sony's EyeToy, Nintendo's Wii Controller and Wii Nunchuk. In addition to these commercial products efforts in creating exergames have been done in academia as well [18, 38]. Sinclair et al. [28] describe considerations which need to be taken into account during exergame creation. In addition to being fun the authors state that game have to have a level of intensity corresponding to the fitness of the player.

Besides being a form of exercise themselves games were also used to encourage both physical activities and healthy eating. In order to promote physical activity, games were used as a requirement for getting more desired activities such as watching TV or playing games [10, 26]. Another strategy involved members of the group competing by using actual data from the pedometer which they were wearing during the day [3, 9]. Examples of games promoting healthy eating include commercial titles like Hungry Red Planet and research ones [34] both of which taught children nutrition skills. A framework for developing serious games can be based on four dimensions: context, pedagogy, learner specifications and representation. This framework can be extended by using output from an analysis phase and checking if the serious game solution satisfies a learning need [5].

GameFlow is a model for evaluating player enjoyment in games [33]. Eight elements ensure a game is fun to play. Concentration is the first one. A game should not distract player from the action and should not burden him with the unimportant tasks. The two elements which are closely

related are challenge and player skill, as they need to be carefully balanced all the time. Next, the player should always feel in control of their actions in the game. Clear goals have to be provided and appropriate feedback should be sent to the player. Element of immersion states that players should be deeply, but effortlessly involved in the game. Social interaction is the last suggested element.

The game described in this paper attempts to combine all the above mentioned concepts, using specialist hardware to exercise, while at the same time promoting healthy eating and physical activity.

4 Game design

Making the game successful required fulfilment of many different, and occasionally opposing, conditions. The game had to be physically engaging, informative and, above all, fun. In order to ensure that the game design is properly executed, guidelines from papers mentioned in previous section were followed [12, 5, 33]. In addition, the fact that the game was non-gender specific and aimed at children aged 8 to 12 years had to be considered. Also, the time available for producing this game was 10 months with a development team including one full time programmer and three part-time members.

The targeted age group governed the choice of the art style, shown in Figure 1, which can be described as cartoony. Characters, environment and even animation are exaggerated with a simple yet vivid colour scheme full of contrast. All edges are accentuated by hard black lines. This style is not only appealing to children but it also accelerates development by reducing time required for modelling, texturing and animation, as less precision and realism is needed. In addition, it should help keep children's concentration as no distraction is created by unnecessary details.

Figure 1: The Game's Art Style

In order to conform to idea of boosting physical activity, the main game input and most of the interaction is carried only through movement. The devices chosen to capture motion were the Wii Controller and Wii Balancing Board, as they are relatively cheap, easily obtainable, provide satisfactory supporting software and given their popularity, a user might already own one. They are used to control both the character and the graphical user interface. The expected player's movement should be bold, thereby, increasing energy expenditure and also making itself easier to capture. This in turn should reduce user errors which should then ameliorate concentration, feeling of control and immersion. Although physical activity is increased in this manner it is acknowledged that the game is not a substitute for real exercise.

The player's character is a young sorcerer (Figure 2). This choice was conditioned by the pedagogical decision of no violence and highly abstracting any notion of combat. The wizard's wand appeared to be a good option for a non-violent tool/weapon. Furthermore, movements made by a wand corresponds well to movements of a Wii Controller making control natural and helping increase immersion. Magic is also convenient way to explain some curious phenomena that happens in a game as will be shown later. In order to preserve gender and ethnic neutrality customization of the character should be allowed.

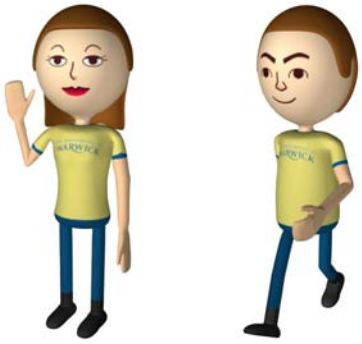


Figure 2: The Main Characters

Level design was guided by the idea of small, independent entities with a number of different yet familiar mini-games. The advantages of this approach are numerous. Firstly, a small, independent level is easily and quickly tested and, in case it does not function, it can be simply discarded. Secondly, the player is not required to complete levels in a predefined order which should increase the impression of control. Likewise, hard levels can be skipped and played afterwards when the player's skill has improved and matched the level's challenge. Lastly, mini-games should increase variety making the game more interesting and keeping the children's attention longer. Still, each mini-game has to have similar gameplay so that the player's learned skill is transferable.

One relatively straightforward solution, which encompasses the suggested type of level design, is to use islands as separate levels. Although, freedom of level selection should be allowed, it would not make sense if player could access the last stage in the beginning of the game. To

overcome this problem and balance challenge and control, the decision was made to have a number of lakes, each containing multiple islands (Figure 3). An island could be freely chosen but transition to the next lake would be locked (Figure 4) until completion of all the levels or scoring a certain number of points. The lakes could have different art themes (e.g. winter, desert, and volcano). In this setting, assets could be reused requiring only slight changes (e.g. textures). This would reduce development time while preserving variety.

Figure 3: One of the Lakes

Figure 4: A Dam Blocking the Exit

The primary vehicle for moving the character is a coracle. This means of transport fits well into the lake environment and interfaces smoothly with Wii Balance Board and Wii Controller. The controller is used for rowing action which propels the coracle forward, while direction is altered by leaning left and right on the Wii Board. This routine, which increases physical activity, has also been used in its own right for a mini-game when the player travels between lakes. To go from one lake to another a player is required to sail down the river full of obstacles (stones, logs, whirlpools) and try to collect bonuses which can later be used on the islands, or increase score (Figure 5).

Figure 5: The River Minigame

Healthy eating messages had to be incorporated subtly, so as not to distract from the main gameplay. The first, simple idea is to place short and relevant text tips onto the loading screen. The second idea, actually, defined the story of the game. As a young sorcerer, the player is required to deliver food packages to the famine struck islanders, while two villain characters (Figure 6), for nefarious reasons, are slipping in unhealthy food in an attempt to make islanders overweight. The player needs to eliminate these unwanted packages. Two pedagogical issues had to be taken into account. Firstly, the main character cannot be modified to look overweight. Secondly, the player is not allowed to fight the villains directly.

Figure 6: The Villains

Taking above considerations into account the gameplay was defined. Upon approaching an island in a lake view the player is transferred onto it. The character still stays in the coracle which now floats above ground (due to magic) but player does not have to row anymore and only uses the leaning motion to move. An island is inhabited by the famished people and animals which add background interest. In the centre of the island is a challenge made from food packages. The ”good” food package is moved into the islanders’ barn when the player collects it, while ”bad” ones are dispatched into a skip. The opposite happens in case of the villains (bad goes to the barn and good to the skip). In this manner direct conflict is avoided. A screenshot pre-

senting the gameplay on one of the islands is shown in Figure 7.

Figure 7: An Island Level

In general, packages are separated into six categories according to food group division (e.g. dairy products, fruit, and vegetables) with one category being unhealthy food (Figure 8). Packages of each group are colour coded, reducing confusion. One of five food group members is represented on a package (e.g. fruit package would be green and could have cherries depicted on it) giving the total of 30 distinct packages.

Figure 8: Blocks Used in a Level

The goal of each level is to provide islanders with enough food packages while achieving good balance of food groups. Preserving this balance during play is important as well, because due to large amount of unhealthy food character starts to turn into stone and his movement becomes slow and lethargic. This also affects islanders and island animals that walk in the background highlighting the consequence. On the other hand, dispatching unhealthy food slows villains down. User interface always provides clear representation of current food balance and time left.

Each island is going to have a different challenge. Central idea behind challenges is the concept of "Physically Enhanced Puzzles". In such puzzles speed and direction of input motion affect the gameplay keeping the player active. Re-interpretations of classic games like Columns, Brake Out, Space Invaders and Tetris are used as a foundation of the puzzle design. They are then modified to accept motion input, collection of food packages and actions of villains. To increase variation, each of these major puzzles is then slightly altered and used on a different island.

5 Game Implementation

The minimal hardware requirements were set low, making the game accessible to more potential users. Also, the low requirements should help deploy the game into the schools with relatively old equipment. The game should run smoothly on a machine with 1GHz processor, 1GB of RAM and 256MB DirectX9 graphics card. In addition, a Bluetooth connection, which is easily obtained using a cheap USB dongle, is required for Wii Controller and Wii Balance Board. Choice for the last two Wii devices is explained in the previous section.

The time frame available for game development ruled out the possibility of creating our own game engine, while budget did not provide funds for purchasing commercially available solution. The only feasible option was usage of a freely available product. Choice was to use Ogre 3D: *"the most powerful open source real-time 3D rendering library currently available"* [15]. It is important to note that Ogre 3D is not a full game engine. It is, instead, a graphics engine used for rendering. However, it is easily scalable with a number of plug-ins provided. Another advantage of Ogre is the large user community which offers solid support and resources. For managing players input from Wii Controller and Wii Board open source libraries were used. Object movement and interaction was implemented using Nvidias PhysX technology.

Game assets and character animation were created in Autodesks Maya 2009 software. Texture production was done in Adobes Photoshop CS2. Both of these packages are industry standards and were available prior to game implementation. Exporting content from Maya to Ogre was done using OgreMax, a freely available Maya plug-in.

6 Feedback

An early version of the game was tested in a local school. Seventeen children aged 7 to 11 played the game. They were of the mixed gender (8 boys and 9 girls), ethnicity and body size. Children were divided into the four focus groups depending on a Year group. A questionnaire was given before playing the game which tested if the children knew they should eat 5 portions of fruit and vegetables a day and also asked them how many portions they ate. The results shown in Table 1 imply that although most of the children knew how many portions they need to eat majority of them did not eat this amount. Children were also

Answer	How many portions of Fruit & Veg should children eat in a day?	How many portions of Fruit & Veg do you eat each day (on average)?
0	0	1
1	0	0
2	1	1
3	1	4
4	1	5
5	12	5
6 or more	2	1

Table 1: Questionnaire Results

asked which computer games they were playing at that moment and 15 of the 17 gave specific names while two participants left this field blank.

16 of the 17 participants were positive about the game with comments like: "funny", "loved it" and "addicted to it". They would like to play the game again. They said they liked the graphics and would be happy to have the game in the school to learn about healthy eating. Some wanted to take a copy home. One girl did not like the game and found it too difficult. She said that it was "annoying".

Children offered other feedback and gave a number of useful suggestions that could be implemented in later versions. X-Box like controller, which was offered during the testing, was preferred over the balance board. As the board was challenging to use children suggested to change its sensitivity. They also said they learned about food. They learned not to eat fat/junk food and connected this with getting fat. Also they showed understanding of different food groups, but sometimes what they said was outside the game's scope (e.g. "different vitamins"). Although there was some mention of physical activity, knowledge which they obtained from the game was limited. The request for customising characters (e.g. clothes, facial features, gender, name and hair) was also expressed. One child suggested tackling the bad guys with swords to which other child responded "that would be violence", although the latter child stated previously he was playing the GTA: Vice City, a rather violent game.

Other suggestions were also brought to researchers' attention and included:

- menu options
- multiplayer option (two players helping each other)
- new levels
- save option
- changing the look of the water in the lakes
- making pictures of food clearer
- clear goals for each level
- option for inputting real-life fruit and vegetable consumption
- option for swimming around the lake
- using the labels of "good food" and "bad food".

In addition to children, four parents/carers participated in the trial. There were 3 mothers and one father, although the mothers gave most of the comments. Parents liked the game and supported the idea of using it in the school to teach children healthy diet. They stated that many cannot afford Wii Balance Board and suggested the usage of the regular controller. They also recommended making the images and text more prominent and clearer. Finally, they said that levels need more feedback, possibly speech, telling children what to do next and giving them useful messages. Parents were keen to see the game going to the next stage of its development.

7 Conclusion

Action to help fight and prevent childhood obesity is required swiftly. It could come in the form of a serious game which increases physical activity and promotes healthy eating and sports. Understanding the background of serious games was important for appreciating the large number of considerations required during development, distinguishing them from regular video games and presenting their advantages. Related work used to govern the design phase was carefully explored. Design was possible only by balancing all different constrains and making sound decisions informed by research. After finishing the design it was possible to start implementation which required new choices in terms of hardware and software. Although the development is still not complete the first trail with a focus group showed that children did enjoy playing the game. However, the true measure of the game's success can only be obtained after the game has been distributed and data, gathered during prolonged use, has been analysed.

8 Future Work

The game is still in the development phase with three months left before the end of the project. Majority of the gameplay logic has been implemented. One instance of fully functional lake with three islands is available. Motion input requires more improvement as there are the occasional problems recognizing gestures. The logic for running cut-scenes is in the late phase of development. The stage for character customization, tutorial levels and nice GUI still need to be created. A strategy for promoting physical activity is also required. There is a present need for generating more content and assets. It is planned to test the game with focus groups once more. After the game is developed it should be deployed and the data to prove its

effectiveness should be gathered, but that goes beyond the scope of this project.

References

[1] BP Bergeron. *Developing serious games*. Charles River Media, 2006.

[2] NHS Information Centre. National child measurement programme: England, 2008/09 school year ONLINE. http://www.ic.nhs.uk/, February 2010.

[3] S. Consolvo, K. Everitt, I. Smith, and J.A. Landay. Design requirements for technologies that encourage physical activity. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, page 466. ACM, 2006.

[4] K. Corti. Games-based Learning; a serious business application. *Informe de PixelLearning*, 2006.

[5] S. de Freitas and S. Jarvis. A Framework for developing serious games to meet learner needs. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, volume 2006. NTSA, 2006.

[6] WH Dietz, LG Bandini, JA Morelli, KF Peers, and PL Ching. Effect of sedentary activities on resting metabolic rate. *American journal of clinical nutrition*, 59(3):556, 1994.

[7] C.B. Ebbeling, D.B. Pawlak, and D.S. Ludwig. Childhood obesity: public-health crisis, common sense cure. *The Lancet*, 360(9331):473–482, 2002.

[8] M.S. Faith, N. Berman, M. Heo, A. Pietrobelli, D. Gallagher, L.H. Epstein, M.T. Eiden, and D.B. Allison. Effects of contingent television on physical activity and television viewing in obese children. *Pediatrics*, 107(5):1043, 2001.

[9] Y. Fujiki, K. Kazakos, C. Puri, I. Pavlidis, J. Starren, and J. Levine. NEAT-o-games: ubiquitous activity-based gaming. In *CHI'07 extended abstracts on Human factors in computing systems*, page 2374. ACM, 2007.

[10] GS Goldfield, LE Kalakanis, MM Ernst, and LH Epstein. Open-loop feedback to increase physical activity in obese children. *International journal of obesity*, 24(7):888–892, 2000.

[11] S.L. Gortmaker. Innovations to reduce television and computertime and obesity in childhood. *Archives of Pediatrics & Adolescent Medicine*, 162(3):283, 2008.

[12] L. Graves, G. Stratton, ND Ridgers, and NT Cable. Comparison of energy expenditure in adolescents when playing new generation and sedentary computer games: cross sectional study. *British Medical Journal*, 335(7633):1282, 2007.

[13] L. Graves, G. Stratton, ND Ridgers, and NT Cable. Energy expenditure in adolescents playing new generation computer games. *British Journal of Sports Medicine*, 42(7):592, 2008.

[14] Serious Games Initiative. Serious games ONLINE. http://www.seriousgames.org/, February 2010.

[15] G. Junker. *Pro OGRE 3D programming*. Apress, 2006.

[16] L.L. Kelly and R.B. Parke. *The pilot maker*. Grosset & Dunlap, 1970.

[17] L. Lanningham-Foster, T.B. Jensen, R.C. Foster, A.B. Redmond, B.A. Walker, D. Heinz, and J.A. Levine. Energy expenditure of sedentary screen time compared with active screen time for children. *Pediatrics*, 118(6):e1831, 2006.

[18] H.H. Lund, T. Klitbo, and C. Jessen. Playware technology for physically activating play. *Artificial life and Robotics*, 9(4):165–174, 2005.

[19] D.R. Michael and S.L. Chen. *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade, 2005.

[20] A. Mitchell and C. Savill-Smith. *The use of computer and video games for learning: A review of the literature*. Learning and Skills Development Agency London, 2004.

[21] C.L. Ogden, M.D. Carroll, L.R. Curtin, M.A. McDowell, C.J. Tabak, and K.M. Flegal. Prevalence of overweight and obesity in the United States, 1999-2004. *Jama*, 295(13):1549, 2006.

[22] M. Prensky. Digital natives, digital immigrants. *On the horizon*, 9(5):1–6, 2001.

[23] JJ Reilly. Obesity in childhood and adolescence: evidence based clinical and public health perspectives. *Postgraduate medical journal*, 82(969):429, 2006.

[24] JJ Reilly, E. Methven, ZC McDowell, B. Hacking, D. Alexander, L. Stewart, and CJH Kelnar. Health consequences of obesity. *British Medical Journal*, 88(9):748, 2003.

[25] L.P. Rieber. Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational technology research and development*, 44(2):43–58, 1996.

[26] BE Saelens and LH Epstein. Behavioral engineering of activity choice in obese children. *International journal of obesity and related metabolic disorders: journal of the International Association for the Study of Obesity (USA)*, 1998.

[27] K.R. Segal and W.H. Dietz. Physiologic responses to playing a video game. *Archives of Pediatrics & Adolescent Medicine*, 145(9):1034, 1991.

[28] J. Sinclair, P. Hingston, and M. Masek. Considerations for the design of exergames. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, page 295. ACM, 2007.

[29] B.K. Smith. Physical fitness in virtual worlds. *Computer*, pages 101–103, 2005.

[30] K. Squire and H. Jenkins. Harnessing the power of games in education. *Insight*, 3(1):5–33, 2003.

[31] C. Stoll. *High-tech heretic*. Doubleday, 1999.

[32] R. Stone. Serious games: virtual realitys second coming? *Virtual Reality*, 13(1):1–2, 2009.

[33] P. Sweetser and P. Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3, 2005.

[34] MC Turnin, MT Tauber, O. Couvaras, B. Jouret, C. Bolzonella, O. Bourgeois, JC Buisson, D. Fabre, A. Cance-Rouzaud, JP Tauber, et al. Evaluation of microcomputer nutritional teaching games in 1, 876 children at school. *Diabetes & metabolism*, 27(4):459–464, 2001.

[35] L. Valenti, J. Charles, and H. Britt. BMI of Australian general practice patients. *Australian family physician*, 35(8):570–571, 2006.

[36] R. Van Eck. Digital game-based learning: It's not just the digital natives who are restless. *Educause Review*, 41(2):16, 2006.

[37] E.A. Vandewater, D.S. Bickham, and J.H. Lee. Time well spent? Relating television use to children's free-time activities. *Pediatrics*, 117(2):e181, 2006.

[38] G.N. Yannakakis, J. Hallam, and H.H. Lund. Comparative fun analysis in the innovative playware game platform. In *Proceedings of the 1st World Conference for Funn Games*, pages 64–70, 2006.

[39] M. Zyda. From visual simulation to virtual reality to games. *Computer*, pages 25–32, 2005.

Methods of simplification for process of 3D animation production

Edin Pašović[1]
Supervised by: Jasminka Hasić [2]
Sarajevo School of Science and Technology
Bosnia and Herzegovina

Abstract

In the modern world, mass-produced computer graphics can be seen almost everywhere, ranging from TV commercials, movies, series, computer games, architecture, archeology, science, history, educational materials, medical research and countless other fields.

Digital animation studios like Pixar and Dreamworks employ thousands of artists and build large render farms and computer clusters for the purposes of rendering CGI (Computer Generated Graphics). Yet, there are thousands of small studios all over the world, producing all sorts of 3D graphics for small-production environments, and they mostly have up to 10 employees and no budget for rendering farms or other expensive computing solutions. They have to focus on using heurestics, shortcuts and alternative rendering methods to achieve results which, despite the low hardware resources for rendering, still keep the look of a high-quality production.

CR Categories: I.3.8 [Computer Graphics]: Applications
Keywords: visual perception, selective attention, GPGPU, parallel processing

1 Visual Language

The way 3D artists create and visualize the scenes themselves can have a great impact on the rendering speed and quality of animations.

A visual language is a set of practices by which images (such as diagrams, professional photos or even motion 3D movies) can be used to communicate concepts and stories. Whether it's a diagram, a professional photograph or a map, they all represent the use of visual language. The structural units of visual language are: shape, line, texture, pattern, color, motion, direction, orientation, scale, space, angle and proportion.[3]

Most known texts on subject of visual language in movies and motion pictures were written by Rudolf Arnheim, a German-born author, art and film theorist and perceptual psychologist. In my project "Greyworld", several of those techniques were used to demonstrate examples of how an efficient usage of visual language can significantly reduce workflow and rendering times.

2 Efficiency in visual language

Recent studies in human perception [Koch, C. and Ullman, 1985; Niebur, E. and Koch, C. 1996; Cater, K., Chalmers, A. and Ward, G. 2003] show increasing success in prediction of eye movements and focus of subjects watching motion pictures and give us further insights into the concepts Rudolf Arnheim first mentioned in his book "Art and Visual Perception: A Psychology of the Creative Eye" in 1954 and later explored in "Visual Thinking" in 1969 and "New Essays on the Psychology of Art" in 1986.

Among many insights into the way the common viewer percieves motion pictures, there is a number of those which, when used in right places, could help us decrease scene complexity or time invested into the creation of the scene in a 3D animation.

Next several examples are used in everyday work in modern animation workflows, and they were not quoted from any particular book, but were acquired by the author of this paper as a result of insight after several years of working in the graphic design and animation industry and listening to the experiences of seniors in that industry. Most of them are also explained in some of the professional DVD tutorial courses [HCWVEFD 2009; HCWMCHEBS 2004].

2.1 Foreground and Background

A large percentage of actual action in motion pictures happens in foreground, thus drawing the focus of the viewer much more on the foreground than on the background.

If there is nothing of importance happening in the background during the scene, the viewer will be

[1] edinpasovic@gmail.com

[2] jh@ssst.edu.ba

gradually less and less focused on the background. This should basically allow us to render the background details with far less detail once we're several seconds into the scene, because by then the viewer will lose interest in analysing the background (like he did for the first few seconds), and he will shift it towards the action in the foreground, not noticing the decline in rendering quality in the background, as long as nothing drastically changes. [HCWMCHEBS 2004]

2.2 Depth of Field

The Depth of Field (DOF) effect, when used inside the 3D software in scenes with high complexity, can increase rendering process up to five times[4].

There is a new method of visualisation, which consists of dividing the scene into several layers, depending on the distance of objects, rendering them without DOF effect, and bringing them together later in compositing software, where DOF simulation will be applied on layers instead of actual objects. While the results are very similar in appearance, the rendering times are significantly decreased[3]. Today this method is mostly being used in the latest generation of game-consoles, such as X-Box 360 and Playstation 3, to simulate DOF in games while maintaining constant frame-rate. [HCWVEFD 2009]

2.3 Moving and static objects

Moving objects tend to draw more attention on themselves than the static objects. Objects that move towards us tend to draw more attention on themselves than the objects that move away from us.

In anthropology this is explained as an old human instinct still left from times when humans used to hunt animals in oder to survive. This behavior can be exploited in order to decide which objects should be modeled and rendered more carefully and which ones require less polygons and simpler rendering procedures, based on their movement. [Glencross M., Chalmers A.G., Lin M.C., Otaduy M.A., Gutierrez D. 2006]

2.4 Color

Human eye is capable of color vision thanks to photosensitive Retinal Ganglion Cells. However, these cells don't percieve all the colors with equal sensitivity.[5] We are most sensitive to red color, less sensitive to green color and even less sensitive to blue color. This is why red and yellow objects (yellow color in RGB color model is combination of red and green) in the scene will draw more attention than the others.

This information can be used to classify which parts of the scene, based on their color, should be given more or less importance in modeling and rendering. [R. Arnheim 1974]

2.5 Composition

Most people in Europe, North and South America, Australia and northern parts of Asia use alphabets written from left to right. This affects their perception so that they observe and analyze static and motion pictures in similar manner - from left to right.

This can be seen in many traditional movie scenes, where camera is following one or more characters from the side during the walking scene - they mostly tend to go from left to right. Video games follow the same analogy, starting from the first "Super Mario" side-scrolling game to the most modern side-scrolling game - characters start at point A on the left side and have to fight their way to the point B which is in most cases located on the right side.

A common viewer tends to focus on what's in front of the character and where the character is going - so in those cases he is mostly focusing on the right side, while the parts of the scene which stay behind recieve much less focus once they are behind characters back. This, again, can be used to gradually reduce the rendering quality of those parts of the scene which are moving towards the left side of the screen. [HCWMCHEBS 2004]

3 Case Study: "Greyworld"

The short 3D animation "Greyworld" was built using both layered 2D and 3D techniques of animation. Since this was a project one single animator was working on, in a limited amount of time (three-and-a-half months), using only one non-workstation computer (Apple iMac, Duo Core 2,6 Ghz, 2 GB RAM, ATI 2660 HD GPU), the goal was to produce an animation while focusing only on important visual details, and invest time and work on areas in regard of how visible and how important they will be to the final audience.

The whole final animation consists of 46 shots distributed through 10 scenes, with a duration of 5 minutes and 40 seconds. During the various production stages, six candidates, with no background or actual involvement in animation or production, were shown segments of the animation and asked if they see any mistakes or details that need additional refining and notices were taken about what they focused on and what they felt was important in the animation.

[4] Actual number depends on the complexity of the scene

5 human eye - ganglion cells." Encyclopædia Britannica. 2010. Encyclopædia Britannica Online. 02 Mar. 2010 <http://www.britannica.com/EBchecked/topic/199272/eye

4 Experiment methodology

Members of the audience were shown single scenes from the animation during the production stage and asked what they find important in the scene and if there are any corrections needed. I always showed them the more complex animation first and the simplified version second.

Questionnaire had the following form:

SCENE A1
Which scene seemed more complex
a) first,
b) second
c) both were equally complex

SCENE A2
Did you notice any difference in fog between the two scenes?
a)yes, clearly
b) yes, but nothing significant
c) not at all

SCENE B
Which model seemed more complex
a) first one
b) second one
c) both were equally complex

SCENE C
Which model seemed more complex
a) first one
b) second one
c) both were equally complex

Answer	a)	b)	c)
SCENE A1	2 (33%)	0	4 (66%)
SCENE A2	0	3 (50%)	3 (50%)
SCENE B	2 (33%)	0	4 (66%)
SCENE C	0	0	6 (100%)

Studying their answers, I realized that they mostly focus on the foreground, paying almost no attention to the background.

They noticed the difference between scenes where I eliminated the background and the scenes where I added it, but they were so focused on the main characters and their actions that they didn't notice the difference between scenes with detailed background and the same scenes with the same background in much lower detail.

Therefore we decided to put a 360 image sphere of an industrial town as a background and skip all the time and work which would be spent into making 3D models

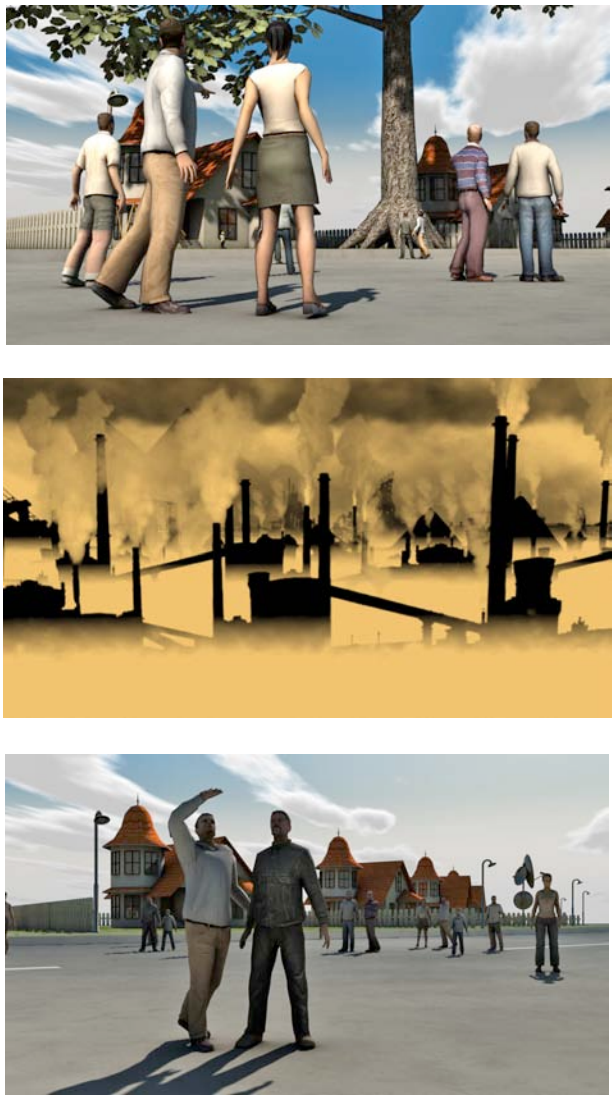


Figure 1: Example frames from the animation "Greyworld"

of background buildings, distributing and rendering them, effectively eliminating several days of work. In order to make this simplification of the background less noticeable, a depth-of-field effect was used to add a slight blur to the background.

There are two main characters, the boy and the robot. Since the boy bears much more similarity with human beings than the robot, audience members were noticing even the smallest mistakes and asked for more details on the model of the boy, when compared to the robot. This is due to the natural human instinct to pay much more attention to detail when watching a human-like being than when observing an animal, robot or a shape. Therefore, the final version of the boy was modeled in 11.000 polygons while the robot didn't take more than 700 polygons.

Also regarding the physical features of the boy's model, viewers paid much more attention to the head and the face than to the other parts and limbs. So in the final model, more than 4.000 of 11.000 polygons were used just to model the face region.

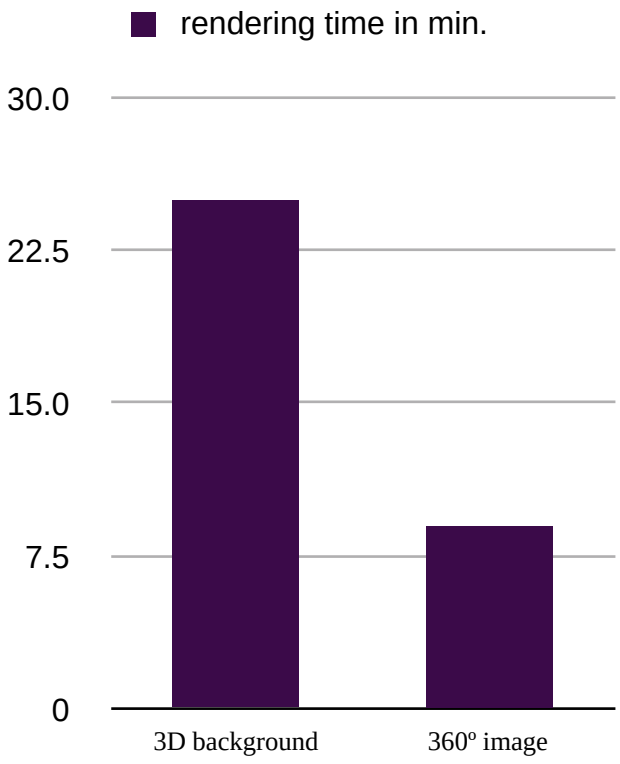
Weather effects and particle effects, such as fog and smoke coming from the chimneys were noticed by the viewers, however they couldn't see any significant difference between 3D volumetric fog and 2D fog made using noise maps in After Effects (a significantly faster method) except in cases where we told them to pay more attention to the fog. The same results came up in the case of smoke from the chimneys in the intro scene. Without being previously warned about the difference between two types of smoke, observers wouldn't notice the difference between the scene where the smoke was simulated through volumetric 3D particles, and the scene where it was just applied as a simple 2D particle effect in post-production stage.

5 Results

Simplification of less important elements of animation led to a noticeable decrease in rendering times.

5.1 Scene A1.

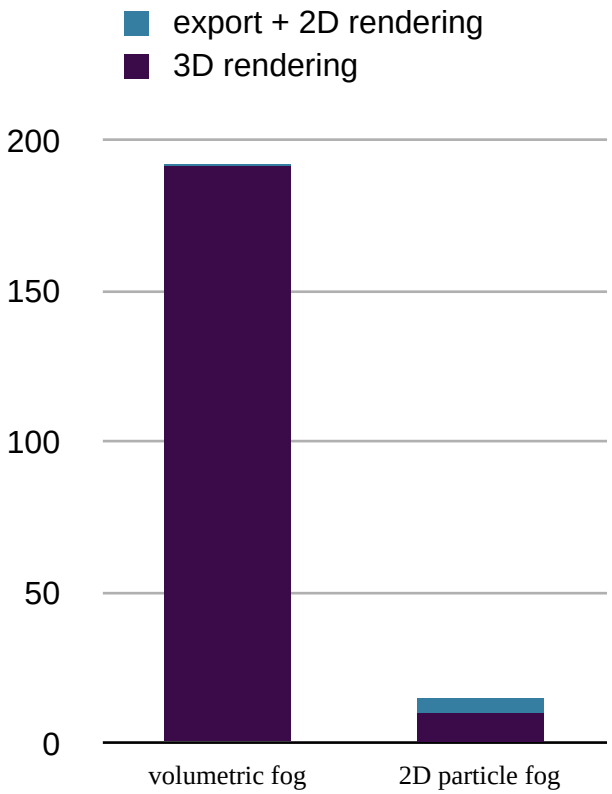
An example scene with 360º image in the background was rendered in 9 minutes, while the same scene with 3D modeled buildings in the background took 25 minutes to render. (only 36% of the original time required). Four of six candidates (66%) noticed no difference before they were told about it.



5.2 Scene A2

In the same example scene, we used volumetric fog, generated in Cinema 4D software, to enhance the mood, and it took over 190 minutes to render, but when we rendered the scene without fog (9 minutes) and exported it to After Effects (in FBX format with 3D cameras and lights), where we added 2D fog with the particle simulation system, it just took another 5,5 minutes for it to render, so the final processing time was 14,5 minutes, compared to 190 minutes with volumetric fog. (7,63% of the original time required).

Three of the candidates (50%) didn't notice the change in the quality of fog before they were told about it, while another three (50%) noticed it but labeled it as not significant in magnitude. But even after noticing the difference in two fog rendering methods, their focus drifted off the fog and to the other elements of the scene. 30 seconds after the fog was introduced into the scene, it was still there and all six of the participants (100%) could confirm its presence, but they were not noticing any details or quality changes regarding the fog anymore, it was just background information now. This led me to conclude that fog could be rendered in full quality and volumetric for the first several seconds of its introduction as an element of the scene, and then lowered in quality and complexity in order to decrease rendering times.



5.3 Scene A3

In this example scene boy is walking across the lawn and then comes to a halt. Two polygonal meshes for the model of the boy were used in two different version of the scene. The complex one consists of 34.000 polygons, the simpler one counts less than 11000 polygons,

optimized so that the most complexity is in the face of the boy.

Six of participants (100%) didn't notice any difference between two versions of the scene while the boy was walking (being in motion). However, two of them (33%) noticed this difference after he came to a halt (stopped moving), but only during the second viewing, after they have been told of the difference. This led me to conclude that simpler versions of the models can be used in moving sequences, while the more detailed (and processor-hungry) models may be left only for still scenes if required.

SCENE C

The same example scene was used as in "SCENE B", however this time I used the 9000 polygon and the 700 polygon model of the robot.

Six of the participants (100%) didn't notice any difference between models in the scenes, and even after being told about it, and after watching still frames from the scene, they couldn't find any considerable difference. This led me to conclude that people pay less attention to detail when watching a non-human creature or robot, than when watching a human or human-like creature.

6 Conclusions and feature work

Using methods such as visual language or experiments with a live audience helps to decide which elements of the scene require a specific level of quality, effectively reducing work and processing times for a significant amount. Further insight in this field could result in classifications, such as a detailed table of specific scene elements showing their relative level of importance and thus saving time and money when working on small and medium-sized 3D animation projects, especially if the animators are less experienced.

While this approach might seem very subjective, with no clear formulas or algorithms, there are still visible patterns in perception of most viewers, which can be statistically measured and compared. Further research could give us answers on how much we have to go into detail with other animation tasks that are heavy on processing requirements, such as fluid animation, cloth dynamics, hair dynamics and other particle dynamics, when working on small-to-medium budget projects, resulting in a clearly defined table of priorities.

Acknowledgements

We would like to thank Andrei Ferko for his professional support during the production stage, Muhamed Kafedzic and Matej Srepfler for sharing their experience on compositing and special effects simplification, and Dino Karadza, Albin Brkic, Dijana Brkic, Elma Selman, Matej Srepfler and Mirza Coric for reviews during the animation production stage.

References

[1] Isaac V. Kerlow. *The Art of 3D Computer Animation and Effects (Third Edition).* Wiley Publishing Inc. 2005.

[2] Rudolf Arnheim. *Art and Visual Perception: A Psychology of the Creative Eye (Second Edition).* University of California Press, 1974.

[3] Glencross M., Chalmers A.G., Lin M.C., Otaduy M.A., Gutierrez D. *Exploiting Perception in High-Fidelity Virtual Environments.* ACM SIGGRAPH, July 2006.

[4] Andrea Di Blas, Tim Kaldewey. *Data Monster - Why graphics processors will transform database processing*. Article on ieee.spectrum website, September 2009.

[5] Rolf Herken. *The Future of Rendering*. NVISION08 presentation, 2008

[6] Rui Wang, Kun Zhou, Minghao Pan, Hu-Jun Bao. *An Efficient GPU-based Approach for Interactive Global Illumination.* Siggraph presentation, 2009

[7] Koch, C. and Ullman, S. *Shifts in selective visual attention: towards the underlying neural circuitry.* Human Neurobiology, 1985.

[8] Niebur, E. and Koch, C. *Control of Selective Visual Attention: Modeling the `Where' Pathway.* Neural Information Processing Systems, 1996.

[9] Cater, K., Chalmers, A. and Ward, G. *Detail to attention: Exploiting visual tasks for selective rendering*. Proceedings of the Eurographics Symposium on Rendering, 2003.

[10] Hasic, J., and Chalmers, A. *Visual attention for influencing the perception of virtual environment.* Spring Conference on Computer Graphics, ACM SIGGRAPH, April 2006.

[11] Hasic, J., and Chalmers, A. *Visual attention for significantly influencing the perception of virtual environments.* Winter School of Computer Graphics WSCG2007 Proceedings, 2007.

[12] NVIDIA Corporation. *Fermi Compute Architecture Whitepaper.* Article on www.nvidia.com

[13] HCWVEFD 2009 - Hollywood Camera Work. *Visual effects for Directors* DVD edition, 2009

[14] HCWMCHEBS 2004 - Hollywood Camera Work. *The Master Course in High-End Blocking & Staging.* DVD edition, 2009

Parallel Distances
Analyzing Multi-Level Relationships in Networks

Stephan Pajer
Supervised by: Harald Piringer

VRVis Research Center
Vienna / Austria

Abstract

This paper introduces a new type of visualization that supports the study of queries concerning relationships between different groups of nodes in a network. It allows a user performing a search for a multi-leveled relationship within a graph to see how each part of the query affects the resulting set of nodes. This view can be efficiently utilized in a linked-view environment to supplement well-established visualizations. The paper finishes off with a case study that exemplifies how to apply the parallel distance view to perform a query on a dataset.

Keywords: Information Visualization, Graph Visualization, Linked Views

1 Introduction

With the recent boom in the popularity of social networks, the importance of extracting information from those networks has undergone a similarly staggering increase. These networks differ from normal network in the distribution of their connections. The maximum distance between nodes is quite small [Wat04] even though most nodes only have a small number of connections [JHGH08]. Additionally, these networks are generally multivariate, which means that there is a multitude of data available for each node of the networks, e.g. age, income, location, etc.

When investigating a social network, one common task users want to accomplish is to search the network for a specific relation between different nodes, each characterized by certain attributes. The nodes can then be grouped together by such attributes as age or income. The user can then create a query to find out how many persons of a group know someone from another group. Such a query might search for persons with a low income that directly know politicians. The query could also be expanded to include more than just two groups. This paper presents 'Parallel Distances', a view that is tailored to assist the user in studying the effects of each part of such a complex relation between groups of nodes in a network.

After briefly presenting related work, this paper will explain this new visualization. We will then go on to explain the interactions we have implemented for this view before describing the necessary computations. The paper continues by giving a usage example of our new visualization before finishing with a discussion and propositions for future work that could be done to improve this view.

2 Related Work

The most widely known visualization of a network is the node-link view. To cope with the size of some networks, several systems that cluster multiple nodes together have been proposed, e.g. [AvHK06] and [AMA08]. This way, node-link views have been successfully utilized in analyzing social networks in [ACJM03] and [PS06]. Others have decided to combine matrix representations [FK46] with node-link diagrams [HF07], [HFM07]. These two visualizations have also been used together in a multi-view environment in [HF06].

Some approaches that also use a node-link diagram to visualize the network but bin the nodes in groups [BMGK08] and [SA06]. Such a binning of multivariate data has also been utilized several times in the recent years to allow the inspection of connections between groups, e.g. [PW06], [PvW08] and [Wat06].

In addition to more traditional network visualization approaches, the Parallel Distance view is also heavily based on Parallel Sets [BKH05] which visualize connections between sets.

3 Parallel Distances

3.1 Visualization

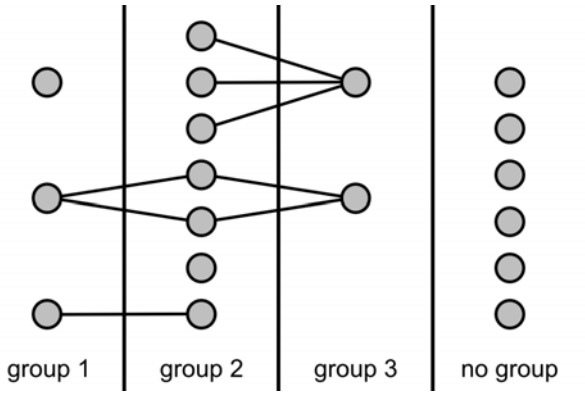


Figure 1: The network visualized in this section.

It should be noted that all figures in this subsection are created from the network pictured in Figure 1. Even though nodes can be members of multiple groups, this example only uses disjunct groups. While all figures are explained in the text, the reader might find it helpful to compare the resulting visualization to the network they are made of.

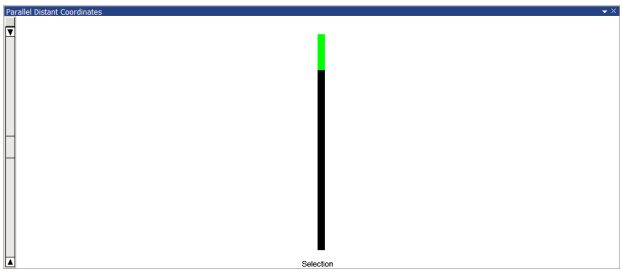


Figure 2: A single axis

First, we will explain how the view looks when just a single group of nodes has been added to the view. An example image of this can be found in Figure 2. A single axis representing the group can be seen. This axis is divided into two parts. The upper part, displayed in green, represents the nodes that are in the group. The lower black part represents the nodes that are not selected. Both parts are scaled according to the number of nodes they represent.

Now we will explain what happens when a second group is added to the view. Figure 3 displays the look of the view with two axes. Two axes is the minimum size for a query to be performed on the groups. Each axis is now divided into 3 parts. While the black part at the bottom has remained unchanged, the top green part has now been divided into two. The red part at the very top shows how many nodes in the group fulfill all requirements of the query. The remaining green part in the middle represents the nodes that are selected in the group but do not meet all requirements of the query.

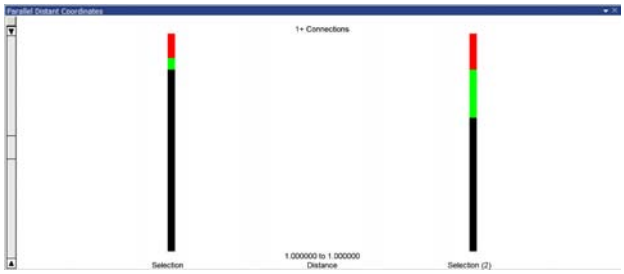
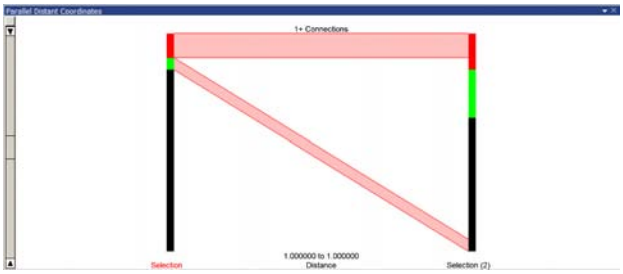


Figure 3: Two axes

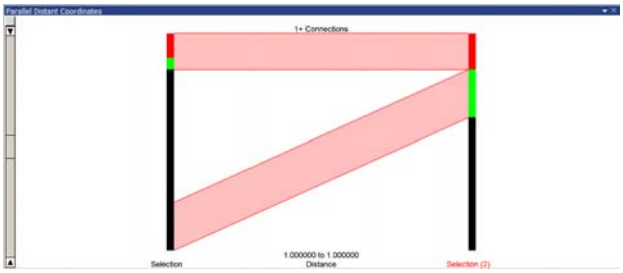
The query is performed from left to right, with a textual representation of the parameters displayed between the axes. Between the axes on the top of the view we display how many connections are required for a node on the left. A node from the group of the left axis that is connected to *at least* this many nodes of the right axis' group is a valid result of this query. Below the number of required connections, on the bottom of the view, we display how large the distance between a node from the left group and a node from the right group has to be. This can be very useful for networks in which each node has a specific weight that represents a value like physical distance between nodes. If, for example, each node represents a city, this value can represent the distance a person has to travel from one city to reach the other one. This way, only cities within a certain distance would be considered for the query.

If the user has selected a node as described in subsection 3.2, the space between the axes is used to show how the connection requirement filters out the nodes that do not fulfill these requirements. See Figure 4 for reference. This is in accordance to the visual information seeking mantra of only offering details on demand [Shn96]. Between the axes are now two connecting segments that represent the nodes in the selected group. Next to the selected axis, both connections are at the top. On the opposite side, one segment stays at the top while another segment points toward the bottom. The segment that stays at the top represents the nodes that fulfill the requirements of the query while the segment that points toward the bottom represents those that do not fulfill the query. Thus, the height of each segment is equal to the respective height on the axis.

Figure 5 shows how the view looks after a third group has been added. To fulfill the query, a node from the left group now has to be connected to the specified amount



(a) Left axis selected

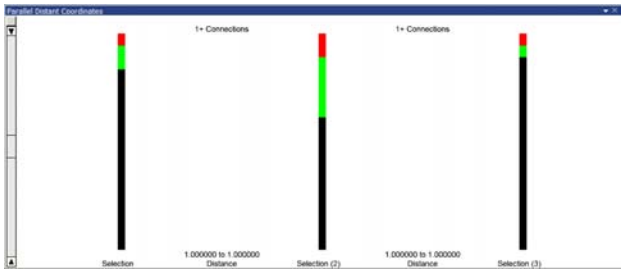


(b) Right axis selected

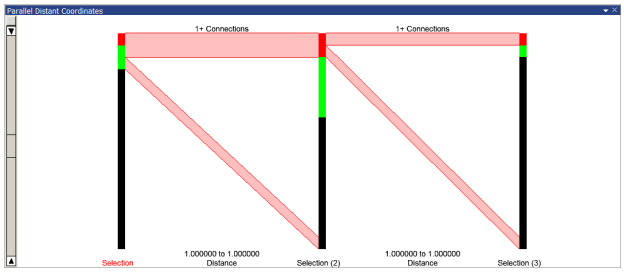
Figure 4: Two axes with one selected

of nodes from the middle group that in turn have to be connected to the specified amount of nodes from the right group. Let us now assume that the user has selected either the left or the right axis. The connecting segments adjacent to the selected axis act as if only the selected axis and the middle axis existed. The connecting segments between the other two axes show the number of nodes that fulfill the whole query staying at the top with the nodes that fulfill the first part of the query but fail the second part point toward the bottom. The size of the segment pointing toward the bottom thus equals the difference between the two segments staying at the top. This allows the user to see which part of the selected group fails the requirements at each part of the query.

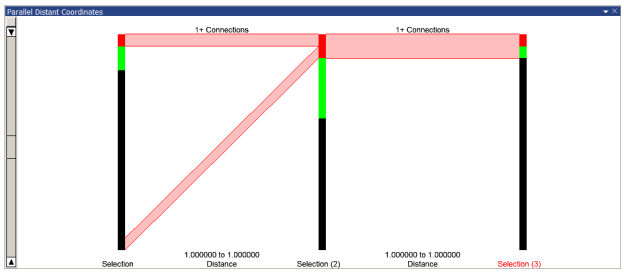
The connections look somewhat different if the user selects the group in the middle. This change is shown in Figure 6. Each connecting segment has been further split into two. The red segment at the very top represents the amount of nodes that fulfill the complete query. The black segment pointing to the top represents nodes that fulfill this connections requirements but fail the requirements between the middle axis and the opposite axis. The black segment pointing toward the bottom stands for nodes that fulfill neither this requirement nor the one from the other side. The second red segment, between the first red segment and the black segment pointing to the top, that points toward the bottom represents the nodes that fail the requirements between these axes but fulfill the requirements between the other two axes. Thus the red segments represent the nodes that fulfill the requirements of the *other* direction while the black segments fail the other direction.



(a) No axis selected



(b) Leftmost axis selected



(c) Rightmost axis selected

Figure 5: Three axes

3.2 Interaction

Since this system is designed to assist the user in exploring the dataset, interaction is very important. First, we will explain how a user can interact with our view, then explain the offered interactions of our view with a linked-view system.

The user may select an axis by clicking on or in the close proximity of it. Selecting an axis updates all connecting segments to reflect the amount of filtered and non-filtered nodes of the newly selected group as described in subsection 3.1.

Besides the main window, an additional control window allows the user to adjust the currently set groups (shown in Figure 7) and also allows the selection of an axis. It also enables to change the order of the axes as well to remove an axis. A dialog to adjust the constraints from the currently selected axis to the next one can also be opened from this control window.

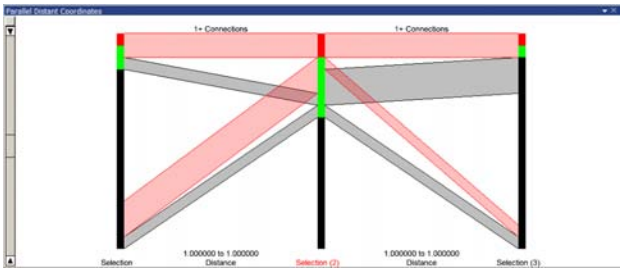
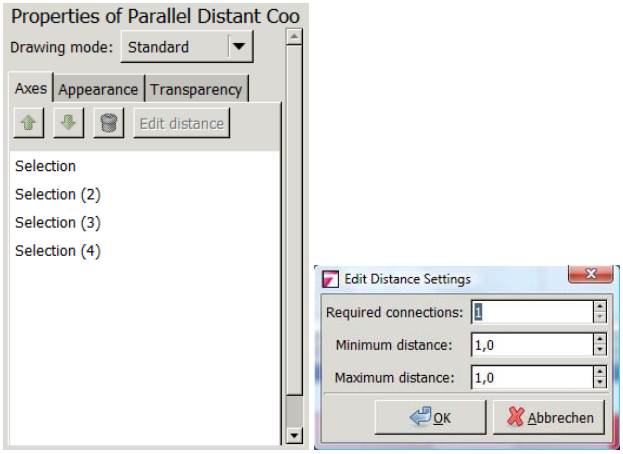


Figure 6: Three axes with the middle axis selected

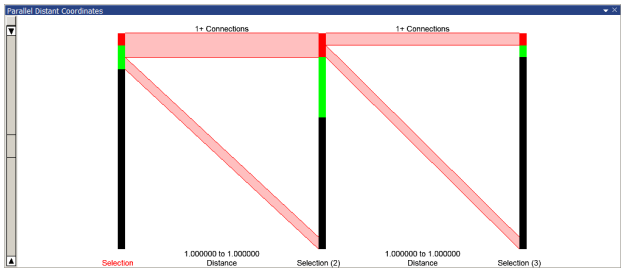


(a) Control window (b) Connection settings

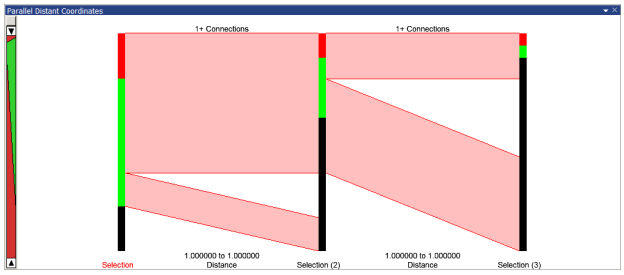
Figure 7: Additional windows and dialogs.

Our view allows the user to zoom into any region he is interested in. Zooming is depicted in Figure 8. This is implemented as a bar to the far left of the window. To zoom, the user simply has to click the widget and drag his mouse. Zoom also affects the displayed connections that originate from this axis. Such a zoom operation is not reset after the user selects a new axis.

There is also interaction between this view and the linked-view system. The user can select any subset of nodes (passes the query, does not pass but is in group, not in group) from any axis. Such a selection affects the whole system and can be utilized in almost any view. Selection also enables the user to transfer the results of a query from this view to other linked views. It is performed by pressing the left mouse button over an axis and highlights the subset of graph nodes that is visually represented below the cursor in all linked views. The framework this view was implemented in supports composite queries based on set operations. This way, selecting the results of the query from all groups is as simple as choosing the unison operation and selecting the desired subset of each distance separately.



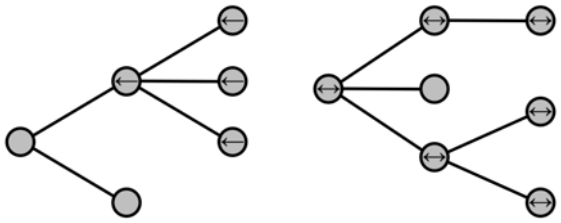
(a)



(b)

Figure 8: The same visualization with (b) and without zoom (a).

4 Involved Query Computations



(a) (b)

Figure 9: Example markings of the algorithm with 2+ connections to the left and 1+ connections to the right.

This section explains the algorithm required to calculate the displayed values of our approach. For this purpose, this algorithm solves the presented query. In addition to the main query, it is also necessary to calculate reduced queries to display the connecting segments for longer (more than 2 axes) queries to display the fraction of nodes that fail the requirements at each axis. These queries differ from the main query in the fact that they ignore axes from one or both sides. For each combination of groups removed from the left and/or right that still has at least two groups left, the algorithm has to run once to solve it. To obtain the number of nodes of a group that fails at a particular distance, the results from the queries with adjacent cutoffs are compared. These results are the same results as the ones that would be obtained from the queries just before and after adding the group. Thus, they

Listing 1 Query Calculation

```
//backward pass
mark all nodes of rightmost axis that are in the group as backward selected
for each segment between axes starting from right
   for each selected node in the left axis
      evaluate connection with backward selected nodes of the right axis
      if the requirements are fulfilled
         mark backward selected

//forward pass
mark all nodes of leftmost axis that are backward selected as forward selected
for each segment between axes starting from left
   for each forward selected node in the left axis
      for each connected and backward selected node of the right axis
         mark forward selected
```

represent the difference in nodes that pass the query this group causes.

The developed algorithm works by performing two passes over every segment between axes. Pseudocode for the algorithm is presented in Listing 1 with an additional illustration showing an example marking in Figure 9. The first pass is a backward pass marking valid nodes starting on the rightmost axis. The second pass starts at the leftmost axis and marks the nodes that pass all requirements and thus fulfill the complete query. For calculations that do not involve all groups in the view, the backward pass information only has to be calculated once per first group on the right and can be reused otherwise.

In the following evaluation, V stands for the number of nodes while S stands for the number of segments between groups (the number of groups minus one). The algorithm has a complexity of $O(V^2)$ for each segment, resulting in a complexity of $O(SV^2)$ for the query and a complexity of $O(S^2V^2)$ for all calculations that have to be performed. In the backward pass, for each segment each selected node on the left side has to be checked against all selected nodes on the right side, resulting in the complexity of V^2. Since S is generally quite small, the major factor influencing the experienced performance of our approach is the size of the groups.

5 Usage Example

This section demonstrates how Parallel Distances can be utilized to analyze a dataset. We utilize our view to perform the search for structure A from the 2nd mini-challenge of the VAST challenge 2009[1]. This challenge asks the user to search the given dataset for an employee at an embassy who communicates with exactly three 'handlers' who pass on his information to a common middle man who then forwards it to the mastermind of a criminal organization. It is further known that the employee has about 40 contacts, each handler has 30–40 contacts, the middle man has 4–5 contacts and the mastermind 'Fearless Leader' has more then 100 contacts including international ones. The handlers are also not allowed to communicate with each other.

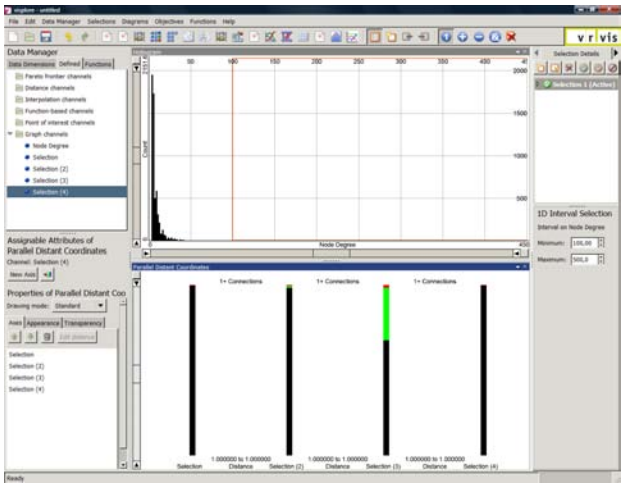


Figure 10: Groups created and added to view.

After loading the data into the system, we start off by defining the groups. We define the employee group as users with 38–42 contacts to incorporate the expressed uncertainty. The handler group is defined as users with 30–40 contacts. It can be seen that these groups overlap. We define the middle man group as persons with 4–5 contacts and add all users with at least 100 contacts to the mastermind group.

[1] http://hcil.cs.umd.edu/localphp/hcil/vast/index.php/taskdesc/index/mini_challenge_2

Next, we open up an instance of our view and add each of those groups in the order employee–handler–middle man–mastermind. A picture of the system at this stage can be found in Figure 10. Now we set the parameters of the connections between the groups. Since a direct connection is the default setting and just the employee needs more than one connected node, we simply set the number of connections between the first two axes (employees and handlers) to three.

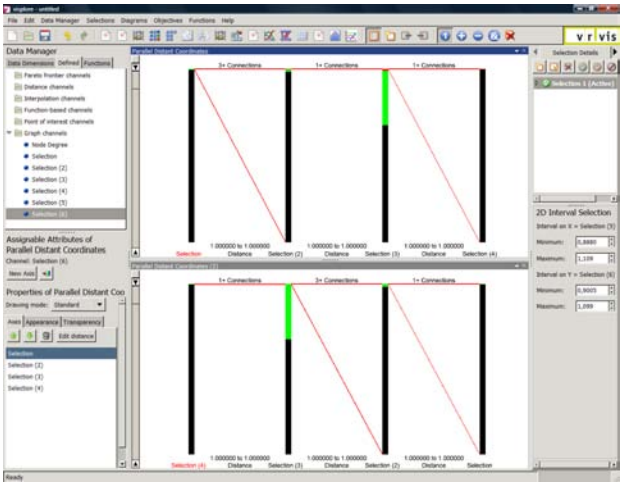


Figure 11: Views and settings needed to solve the problem.

The only requirement we have not yet incorporated is that all three handlers need to be connected to a single middle man. To specify this requirement, we add a second instance of our view to the system. This instance also contains all groups, but in reverse order: mastermind—middle man—handler—employee. We set the required number of required connections between the second and third axis (middle man and handler) to three. This is shown in Figure 11.

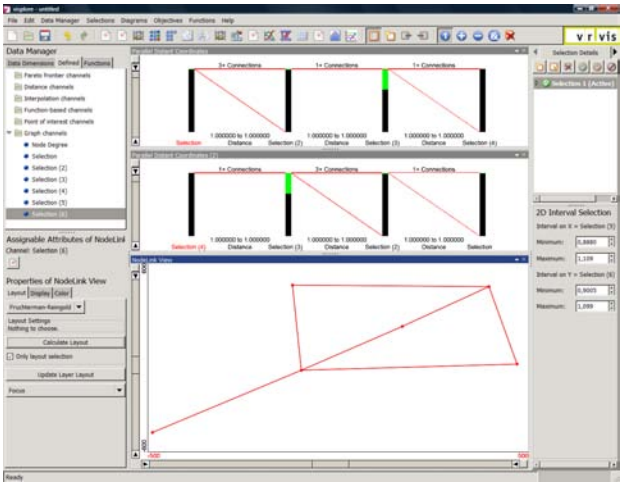


Figure 12: Visualization of the result.

An intersection between the same groups from the two views now yields the nodes that fulfill the requirements of both views. Since there could still be nodes that fulfill the requirements of the two views with different neighbor nodes, we add a node-link view to the system and visualize the results. Such nodes can be easily identified in such a view, since such neighbor nodes are already filtered out and the resulting structure does not match the requested one. In fact, such nodes will most probably have no neighbors or just a single neighbor with no further connections after such filtering. Finally, we still have to verify that the handlers do not communicate among themselves. Thankfully, there are only six nodes not filtered out, and they match the requested layout, so those steps can be skipped. The final state of the system can be seen in Figure 12.

6 Discussion and Future Work

While the presented view succeeds in offering the user the intended information, there are still a number of points that could be improved. As described in section 5, it is currently not possible to specify that a node needs to be connected to multiple nodes that are further all connected to the *same* node (i.e. a person in group A knows two persons in group B that *both* know the *same* person in group C). In the current version, this would require two views where the second instance contains the same groups as the first one but in reverse order. It would further require the user to check the result for fragments of solutions where one part managed to fulfill all requirements while the nodes used to fulfill them did not pass. However, it is easily possible to imagine a constellation that can not be answered with two instances of this view by requesting such a constellation multiple times in the same query. A possibility to exactly specify the required layout would be a significant improvement for this visualization.

Another extension concerns the support of fuzzy groups. If the given group values would be treated as a percentage of group membership instead of a binary value, the user could dynamically adjust the threshold and thus alter the group membership (i.e. for a group defined by age, the user could see if reducing the threshold by a single year would cause a significant change in the result of the query). This would reduce the time required to modify groups significant. It would further be possible to change the binary group membership to show one or more colored sections that reflect how close these sections are to fulfilling the group requirements. This would enable the user to assess the impact of changes to the group membership.

7 Conclusions

This paper presented a novel view for systems that utilize multiple linked views with the goal of giving users that are interested in searching for complex relations within a multivariate network using a multiple linked view environment. Our approach supports the study of queries concerning relationships between groups in a network. To further investigate the results of such queries, this view has been integrated into a system containing views that can be used to further investigate the results. A useful combination with our view that is available in this system is in the form of an already existing node-link view. This combination has proven to be effective for both small networks and cases where the user already has at least a basic idea of the kind of relations he is interested in investigating.

8 Acknowledgments

This work was done at the VRVis Research Center in Vienna, Austria. Special thanks go to my supervisor Harald Piringer as well as Wolfgang Berger for the discussions we had while planning the view.

References

[ACJM03] AUBER D., CHIRICOTA Y., JOURDAN F., MELANÇON G.: Multiscale visualization of small world networks. In *INFOVIS* (2003), pp. 75–81.

[AMA08] ARCHAMBAULT D., MUNZNER T., AUBER D.: Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 900–913.

[AvHK06] ABELLO J., VAN HAM F., KRISHNAN N.: Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics 12* (2006), 669–676.

[BKH05] BENDIX F., KOSARA R., HAUSER H.: Parallel sets: Visual analysis of categorical data. In *INFOVIS '05: Proceedings of the 2005 IEEE Symposium on Information Visualization* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 133–140.

[BMGK08] BARSKY A., MUNZNER T., GARDY J., KINCAID R.: Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE Transactions on Visualization and Computer Graphics 14* (2008), 1253–1260.

[FK46] FORSYTH E., KATZ L.: A matrix approach to the analysis of sociometric data. *Sociometry 9*, 4 (1946), 340–347.

[HF06] HENRY N., FEKETE J.-D.: Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 677–684.

[HF07] HENRY N., FEKETE J.-D.: Matlink: Enhanced matrix visualization for analyzing social networks. In *Proceedings of the International Conference Interact* (2007), pp. 288–302.

[HFM07] HENRY N., FEKETE J.-D., MCGUFFIN M.: Nodetrix: Hybrid representation for analyzing social networks, 2007.

[JHGH08] JIA Y., HOBEROCK J., GARLAND M., HART J.: On the visualization of social and other scale-free networks. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1285–1292.

[PS06] PERER A., SHNEIDERMAN B.: Balancing systematic and flexible exploration of social networks. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 693–700.

[PTMB09] PIRINGER H., TOMINSKI C., MUIGG P., BERGER W.: A multi-threading architecture to support interactive visual exploration. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1113–1120.

[PvW08] PRETORIUS A. J., VAN WIJK J. J.: Visual inspection of multivariate graphs. *Comput. Graph. Forum 27*, 3 (2008), 967–974.

[PW06] PRETORIUS A. J., WIJK J. J. V.: Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 685–692.

[SA06] SHNEIDERMAN B., ARIS A.: Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 733–740.

[Shn94] SHNEIDERMAN B.: Dynamic queries for visual information seeking. *IEEE Softw. 11*, 6 (1994), 70–77.

[Shn96] SHNEIDERMAN B.: The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96: Proceedings of the 1996 IEEE Symposium on Visual Languages* (1996), IEEE Computer Society, p. 336.

[Shn97] SHNEIDERMAN B.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Wat04] WATTS D. J.: *Six Degrees: The Science of a Connected Age*. W. W. Norton & Company, February 2004.

[Wat06] WATTENBERG M.: Visual exploration of multivariate graphs. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 811–819.

Modeling and Natural Phenomena

Extraction of skinning data by mesh contraction with Collada 1.5 support

Martin Madaras*
Supervised by: Tomáš Ágošton†

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

Abstract

The most common approach to animate models and determine their shape attributes in computer graphics is using skeletons. The skeleton and skinning weights can be either assigned manually or computed from an input mesh. This paper proposes the extraction of a skeleton and skinning weights from a mesh, describes how to store computed data in Collada 1.5 and use it for an animation. Firstly, the mesh is contracted using constrained Laplacian smoothing in a few iterations. Then the most important vertices from the contracted mesh are chosen as control points. Multiple edges are removed and vertices that are very close to each other are merged. We select and collapse a vertex pair with the minimum cost in every iteration using a greedy algorithm. The greedy selection is applied repeatably until we have the requested number of bones. In the next step the skinning weights are computed, according to if we want rigid or soft skinning. In the postprocessing stage the user can inspect the skeleton by previewing skinning deformations, make desired changes and export the skeleton to Collada 1.5. Transformation matrices used in a hierarchical skeleton tree are not transformed to joint's local transformation frame, so they are immediately compatible with majority of animation software and libraries. After the Collada file with the mesh, the skeleton and skinning data is exported, data can be imported in animation software such as 3D Studio Max, Blender or Maya and a skinning animation can be rendered.

Keywords: skeleton extraction, mesh contraction, Collada 1.5, skinning

1 Introduction

A frequently used approach for animation and modification of 3D models is based on creating articulated hierarchical structures - skeletons. Skinning data as a skeleton tree and weights can be either assigned manually or computed from an input mesh. The first option is most often chosen by artists, although sometimes it is unnecessary and time consuming. The skeleton has to be created (or imported from templates), rigged into the mesh and influence weights have to be set. Skilled artists are able to create and rig the skeleton in a short time, but sometimes they have to make a lot of rigging adjustments during the skinning process. In this paper we present how to automatically compute the hierarchical skeleton and skinning weights from an input mesh and use them in a skinning animation using Collada 1.5 as export format between our application and a graphic animation software such as 3D Studio Max, Blender or Maya. Our application also provides a way how to examine the computed skeleton before exporting. The skeleton can be inspected by applying skinning deformations using direct kinematics. The interface also allows to dynamically add or remove a bone or a branching if the user thinks some changes are needed.

2 Related work

2.1 Skeleton extraction

Numbers of algorithms have been proposed to compute a skeleton from the mesh geometry. There are three main groups of algorithms: volumetric methods, example based methods and geometric methods. In this paper we focus on geometric methods, they are the most suitable for meshes, because there is no conversion needed. The geometric methods work directly on polygon meshes. The most widely used geometric methods are Reeb graph based methods [2], Voronoi diagram based [7] and Laplacian smoothing based methods [1].

Reeb graph based methods need a suitable real-value function, defined on the model surface, for a successful extraction of a skeleton. Using this function, nodes of 1D graph can be computed. This graph encodes topology of the mesh and after resampling it is used as a base for the skeleton. The method based on a harmonic function proposed by [2] captures after resampling all the features of the model well, but requires the user to specify the boundary condition explicitly.

Laplacian smoothing based methods work directly on

*martin.madaras@gmail.com
†tomas.agoston@abyss-studios.sk

the mesh geometry. The main idea of this approach is to apply a well defined filter on mesh vertices. These methods solve the Laplacian system with different weights to constrain the global smoothness and the volume preservation.

A few more approaches to the skeleton extraction problem are worth to mention. [13] extract skeleton by simplifying the Voronoi skeleton with a small amount of user assistance. [11] use repulsive force fields to find a skeleton. The problem has received a lot of attention in recent years and yet the design of a simple and robust method for extracting curve-skeletons remains a research challenge [5].

2.2 Skinning

Many types of mesh deformations can be performed by a skeleton-driven deformation. However, there are types of mesh deformations such as wrinkles, skin folds and another non-bone-driven deformations, where skeleton-driven deformation is not a sufficient option. Examples of non-bone-driven deformation methods are surface based methods [10, 15] and volume based methods [16]. Unfortunately, these methods are not suitable for a real time animation of high resolution meshes in present. Because of its efficiency and simple GPU implementation the most popular skeleton-driven method still remains linear blend skinning (LBS), also known as skeleton subspace deformation. Some real time skinning works have focused on improving the LBS by inferring the character articulation from multiple meshes.

A few solutions to the problem of finding skinning weights were proposed [3], but the methods are either resolution dependent [9] or the weights do not vary smoothly along the mesh [14], causing artifacts with high resolution models.

3 Graph conversion

For running our graph algorithms, we need to have the input mesh as one connected object. The object needs to be converted into a 3D graph, defined by an edge matrix E. It is quite common that models are composed of more objects. These objects appear to be connected visually, but edges in the model structure between these objects are not defined - Figure 1. Also the opposite problem has to be considered. There can be edges defined in an input mesh which connect parts that should not be connected - Figure 2. These edges are remains of the work of graphic designers or artifacts after format conversion and therefore have to be excluded.

3.1 Joining and splitting of objects

Using a simple depth-first search suitable joining distances can be found to connect or disconnect all graph components. The algorithm works in two phases. First, we construct a mesh graph from an input mesh. This mesh graph is constructed in a straight-forward way from the original model structure and may consist of components. In the next step, we compute distance between each pair of components. For each component, the joining distance is computed as a minimum of distances to each other component. In the second phase, we construct the mesh graph again, using joining distance tolerance computed in first phase. This means that each vertex is joined with vertices which lie in the joining distance radius. This condition joins the closest vertices in neighbouring components which creates a one-component graph. An opposite approach can be used when we want to avoid cycles in the final skeleton. We can either compute or manually set splitting distance tolerance. All the edges with a distance smaller than this tolerance, will be removed.

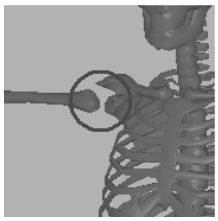


Figure 1: Joining of the mesh graph is needed.

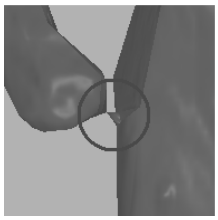


Figure 2: Splitting of the mesh graph is needed.

4 Mesh contraction

For contraction of the generated mesh graph we use the contraction algorithm using Laplacian smoothing proposed by [1]. The algorithm does not alter geometry connectivity (final skeleton curve is homotopic to the original mesh), is noise sensitive and works directly on the mesh geometry (the model does not have to be resampled). Geometry contraction removes details from the surface by applying Laplacian smoothing.

4.1 Laplacian smoothing

Vertex positions are smoothly contracted along their normals by solving the equation (2). The Laplacian smoothing operator (1) was introduced by [6] for surface smoothing. Laplacian smoothing operator L is the $n \times n$ square matrix. This operator is applied on n vertices in vector V as a filter. Term LV approximates curvature flow normals,

so solving $LV' = 0$ removes normal components of vertices and contracts the geometry, resulting into a new set of vertices V'.

$$L_{ij} = \begin{cases} w_{ij} = \cot\alpha ij + \cot\beta ij & \text{if } (i,j) \in E \\ \sum_{(i,k)\in E}^{k} -w_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where:

E – is the set of edges defined in the previous section during a graph conversion process

$\alpha ij, \beta ij$ – are the opposite angles corresponding to the edge (i,j) [6]

4.1.1 Linear equation

Unconstrained solving of this equation contracts the mesh graph into a single point, so the equation is solved in more iterations with carefully chosen weights which control the contractions. W_L and W_H are diagonal weighting matrices which control the contraction process. Weighting matrices have to be updated after each iteration to drive the iteration process into a desired state. By increasing $W_{L,i}$ we can increase the collapsing speed for vertex i and by increasing $W_{H,i}$ we increase the attraction weight to attract vertex i to its current position. All the $W_{L,i}$ are in the next step multiplied by a predefined constant (s_L) and $W_{H,i}$ are updated in such a way, that the attraction weight is multiplied by a ratio of the change of the area of faces adjacent to the vertex i. If the area of adjacent faces is smaller, the multiplicative term is higher, vertices are more attracted into their current positions and in the next iteration the geometry is less contracted in these vertices.

$$\begin{bmatrix} \mathrm{W}_L L \\ \mathrm{W}_H \end{bmatrix} V' = \begin{bmatrix} 0 \\ \mathrm{W}_H V \end{bmatrix} \quad (2)$$

Each step of the iterative contraction process works as follows (t denotes the iteration number):

1. Solve $\begin{bmatrix} \mathrm{W}_L^t L^t \\ \mathrm{W}_H^t \end{bmatrix} V^{t+1} = \begin{bmatrix} 0 \\ \mathrm{W}_H^t V^t \end{bmatrix}$
2. Update $W_L^{t+1} = s_L W_L^t$ and $W_{H,i}^{t+1} = W_{H,i}^0 \sqrt{A_i^0 / A_i^t}$, where A_i^0 and A_i^t are the original and current areas of adjacent faces for the vertex i, respectively.
3. Compute the new Laplacian operator L^{t+1} with the vertex positions computed from the previous iteration V^{t+1} using equation (1).

The iterations converge when the volume is close to zero. After each iteration, the volume approximation has to be computed. In our implementation we used an approximation algorithm which subdivides the bounding box of the model into an octree structure.

Figure 3: The half-edge collapse $(\tilde{v_2} \to \tilde{v_1})$.

5 Skeleton construction

The contracted mesh graph from the last iteration is simplified, very close vertices are merged and a greedy algorithm is used to select the most important control points. In this section we are going to work with already contracted vertices from previous section, they will be denoted as $\tilde{v_i}$. In our implementation the user can choose, how many control points he wants. We used 24 control points by default which worked well for almost all GPUs and also it is a sufficient number to control the skinning process of complex high resolution models. During the collapsing process, for each control point, the collapsed vertices into this control point are stored in a hash map. Vertices are merged into control points and every control point is shifted into the center of its local mesh area which can be computed from the stored hash map. For each control point, the volume of the mesh region the control point represents in the original mesh is computed. The control point which represents the largest volume is chosen as the skeleton root.

5.1 Mesh graph simplification

The first step in the mesh graph simplification is to collapse all edges, whose vertex distance is smaller than a predefined threshold. Vertices in such pairs can be interpreted as the same control point, because the distance between them is very small and the influence over the mesh vertices is almost the same. The second step is to collapse edges which are the least important. For every edge the cost value is computed and edges with minimum cost are collapsed. For simplicity we apply half-edge collapse. The half-edge collapse $(i \to j)$ merges vertex i to vertex j and removes all the faces that are incident to the collapsed edge. The half-edge collapse $(\tilde{v_2} \to \tilde{v_1})$ is shown in Figure 3. This step is repeated so many times that in the end we will have the desired number of control points. The cost value is computed as a weighted sum of a sampling cost term and a shape cost term.

The sampling cost term (3) penalizes collapsing which generates long edges, because in that way we will loose the good mapping between the skeleton and the surface. The term is computed as a weighted sum of distance of adjacent vertices to the collapsed vertex.

$$\mathrm{SCT_a}(i,j) = \|\tilde{v_i} - \tilde{v_j}\| \sum_{(i,k)\in\tilde{E}} \|\tilde{v_i} - \tilde{v_k}\| \quad (3)$$

where:

$\tilde{E}$ – is the current simplified edge set

The shape cost term (4) works in almost the same way as in QEM simplification method [8] with one change. The error matrices are computed over the edges, because the contracted mesh has zero area faces, so the original volume based approach cannot be used. A symmetric 4×4 matrix Q is associated with every vertex. Q is defined in a such way, that term $\mathrm{F_i}(p) = p^T Q_i p$ is a squared distance between point p and the edge (i, j). The initial error matrix for vertex i is the sum of all squared distances to its adjacent edges. For more detailed description of these matrices, their initialization and their use in calculation of the shape cost term we refer to [1]. For a given contraction $(\tilde{v_i}, \tilde{v_j})$ a new matrix Q needs to be derived to approximate the error at $\tilde{v_j}$. Error matrices from previous iterations are stored, so each cost update involves only matrix addition. The shape cost term guarantees to keep the shape of the contracted mesh graph as undisturbed as possible during the simplification. The idea to assign these cost terms to each edge after the iterative contraction converges successfully origins from [1].

$$\mathrm{SCT_b}(i, j) = \mathrm{F_i}(\tilde{v_j}) + \mathrm{F_j}(\tilde{v_j}) \tag{4}$$

6 Binding skin vertices

Once we get the skeleton, we bind the mesh vertices to its joints. If we attach a rigid model, the skin is supposed to be inflexible. Therefore we only anchor a mesh vertex to one nearest control point. In other way, when we want to animate a character, we want the vertices to transform smoothly. In this case, mesh vertices have to be anchored to more control points with corresponding weights.

6.1 Skinning weights

Skinning indices are computed by finding a set of closest control points to each vertex. The geodesic distance is used as a distance measure. A distance between each pair of vertices on the mesh graph from $0th$ iteration (after conversion from an input mesh) is calculated and stored in matrix D. It is calculated using Floyd-Warshall algorithm [4], before the mesh graph is contracted. For each control point C_k, the closest mesh graph vertex is found, for instance v_j. Then, the resulting geodesic distance (5) between the control point C_k and the mesh vertex v_i is computed as a sum of distance between v_j and v_i on the mesh graph calculated by Floyd-Warshall algorithm and the euclidean distance between C_k and v_j. The illustration is shown in Figure 4.

$$\mathrm{gd}(i,k) = \mathrm{D}[i, j] + \mathrm{d}(C_k, v_j) \tag{5}$$

where:

$\mathrm{d}(C_i, v_k)$ – is euclidean distance between the mesh vertex v_j and k^{th} control point C_k

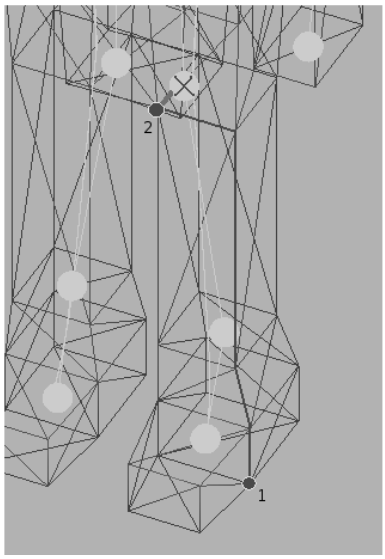


Figure 4: This figure shows computation of the geodesic distance between the control point marked with red cross and the mesh vertex (1) on the bottom right. The red path is the shortest path on the mesh calculated by Floyd-Warshall algorithm and the blue line is the euclidean distance between the selected control point and the closest vertex (2) to this control point on the mesh.

Weights (6) are assigned in a way that weight sum for each vertex is equal to 1.0. Fractions are constructed in a way that weights are indirectly dependent on the geodesic distance. This construction guarantees that the closer control points will have greater influence over mesh vertices than the further ones. The geodesic distance is a real-value function defined on the mesh surface. Because the function varies smoothly along the mesh, the resulting weights are fluently distributed over the mesh regions.

$$\mathrm{weight}(i,k) = \frac{\frac{1}{\mathrm{gd}(i,k)}}{\sum_{k' \in S} \left(\frac{1}{\mathrm{gd}(i,k')} \right)} \tag{6}$$

where:

$\mathrm{gd}(i,k)$ – is geodesic distance between the mesh vertex v_i and k^{th} control point C_k

S – is the set of control point indices controlling the vertex v_i

Floyd-Warshall algorithm has time complexity $O(n^3)$, so it takes quite a long time on models with higher number of vertices. To optimize that time, we can use a downsampled mesh for this computation. For downsampling, we used previously mentioned QEM simplification method [8]. The downsampled mesh preserves the mesh branching, tunnels and important vertices.

6.2 Joint matrices

Bind pose matrices and current transformation matrices for all the nodes are stored in the global (root local) space. They are not transformed into node's local space. The disadvantage is that during the skinning preview the matrices have to be transformed into node's local space. On the other hand, the main advantage is that the skeleton structure is compatible with the majority of animation software.

6.3 Skinning on GPU

Our framework provides linear blend skinning implemented on GPU for real-time examination of computed skeletons. It is the most suitable method how to inspect the skeleton structure and data, because of its efficiency and simple GPU implementation. After the skeleton is computed, the "bind pose" world-space snapshot of all transformation matrices of the skeleton nodes is taken, denoted as B_i, for each skeleton node. During the real-time deformation process, transformation matrices are computed. Each transformation matrix, storing the current affine transformation, denoted as P_i, is computed each time the user manually changes skeleton nodes. In each frame, the resulting transformation M_i is computed as $M_i = P_i B_i^{-1}$ on CPU and uploaded into GPU. New deformed vertices are computed using GLSL shader as :

$$v' = \sum_{i=0}^{n} w_i M_i v_i \tag{7}$$

where:

M_i – is the resulting transformation matrix, computed on the CPU as $M_i = P_i B_i^{-1}$

w_i – is the associated weight

v_i – is the original vertex position in the M_i coordinates system

6.4 Robustness

The mesh contraction and the skeleton extraction phases are pose independent. It enables extraction of compatible skeletons from different poses of the same model. By compatible, we mean compatible in the sense of the same branching, tunnels and the homotopy with an input mesh. Also, the length of preserved edges will be the same, because edges are collapsed in the same order. In the end of the skeleton construction process, all corresponding skeleton bones will have the same lengths. The geometry meshes in different poses have corresponding edges of the same length as well, so the skinning weights and indices will result into identical values.

7 Collada 1.5 support

Model data and all important skinning data is saved in a compatible way into the Collada *.dae* XML file. Collada is a Collaborative Design Activity for establishing an interchange file format for interactive 3D applications. Collada supports storing of the mesh geometry, the hierarchical skeleton structure, indices, weights and inverse bind pose matrices. With version 1.5 comes the possibility to store kinematics as well. To use the full support of a kinematics model in Collada 1.5, computation of approximation of the minimum, the maximum and the current angle for each joint is needed. Angles can be computed using the inverse kinematics approach or assigned manually by the user. In our implementation we use the latest version of Collada DOM 2.2 with Collada 1.5 [12] support.

8 Results

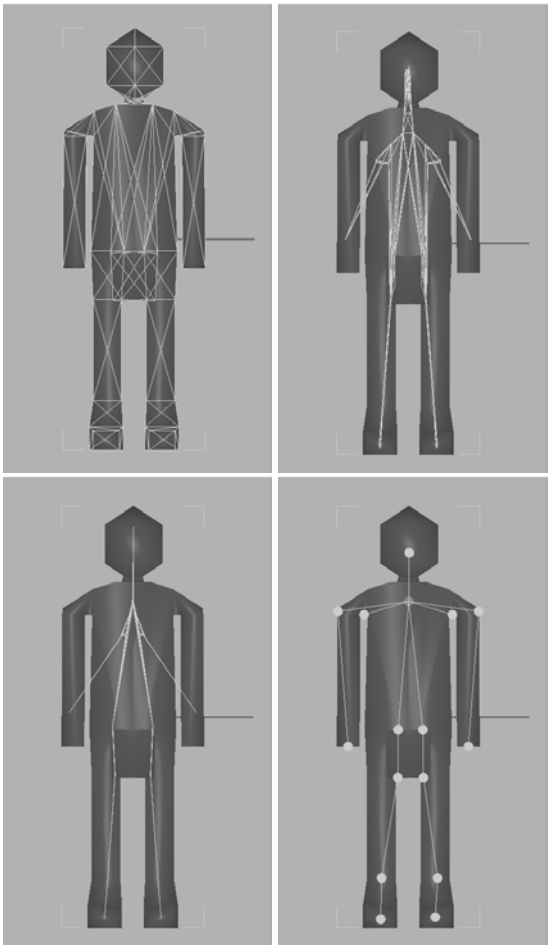


Figure 5: The contraction and the extraction of the skeleton from low resolution geometry. (Top-left) The converted mesh graph. (Top-right) The mesh graph after 2 iterations. (Bottom-left) The mesh graph after the last iteration, the volume approximation is close to zero. (Bottom-right) The extracted skeleton.

We have tested the framework on a wide range of different models. We have achieved good results with both high resolution and also with low resolution models. The more

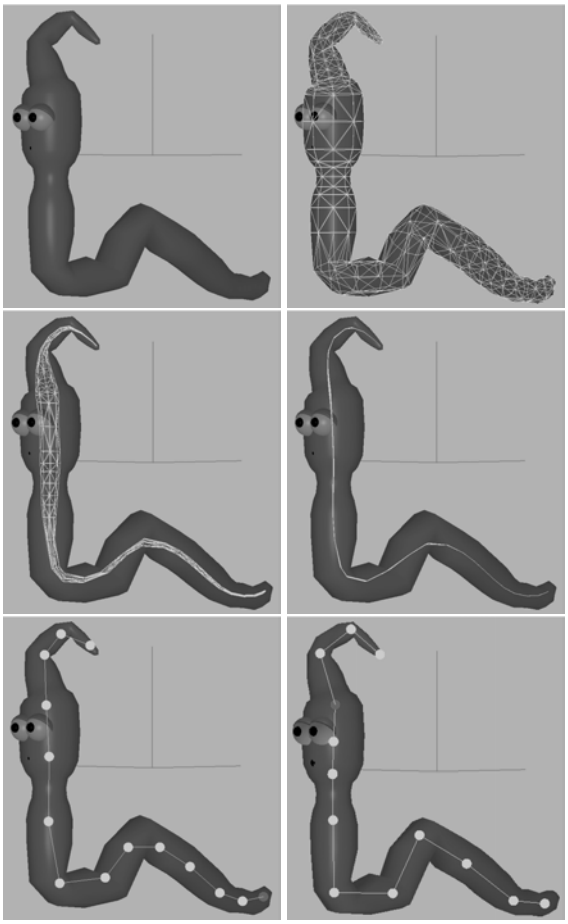


Figure 6: The contraction and the extraction of the skeleton in few iterations from higher resolution geometry and its comparison to manually rigged skeleton by an artist. (Top-left) An input model. (Top-right) The converted mesh graph. (Middle-left) The mesh graph after 2 iterations. (Middle-right) The mesh graph after the last iteration. (Bottom-left) The extracted skeleton. (Bottom-right) The skeleton rigged by an artist.

vertices the model has, the better skeleton can be computed, but the algorithm takes more time. Also the low resolution models (less than 5000 polygons) can be contracted in a good way, but it is harder to set good contraction weights. Setting the right contraction weights is the most problematic part of this approach. Low resolution models are more sensitive for high curvature differences and can be easily over-contracted. Weights often have to be set manually and the user needs some experience. Contraction of high resolution models is more deterministic and good weights can be set automatically. Contraction of geometry with lower number of vertices can be seen in Figure 5 and contraction of geometry with more vertices can be seen in Figure 6 and 7. The model of a worm in Figure 6 was published with a manually rigged skeleton (Bottom-right image). After the comparison we can conclude that the extracted skeleton is very close to the manu-

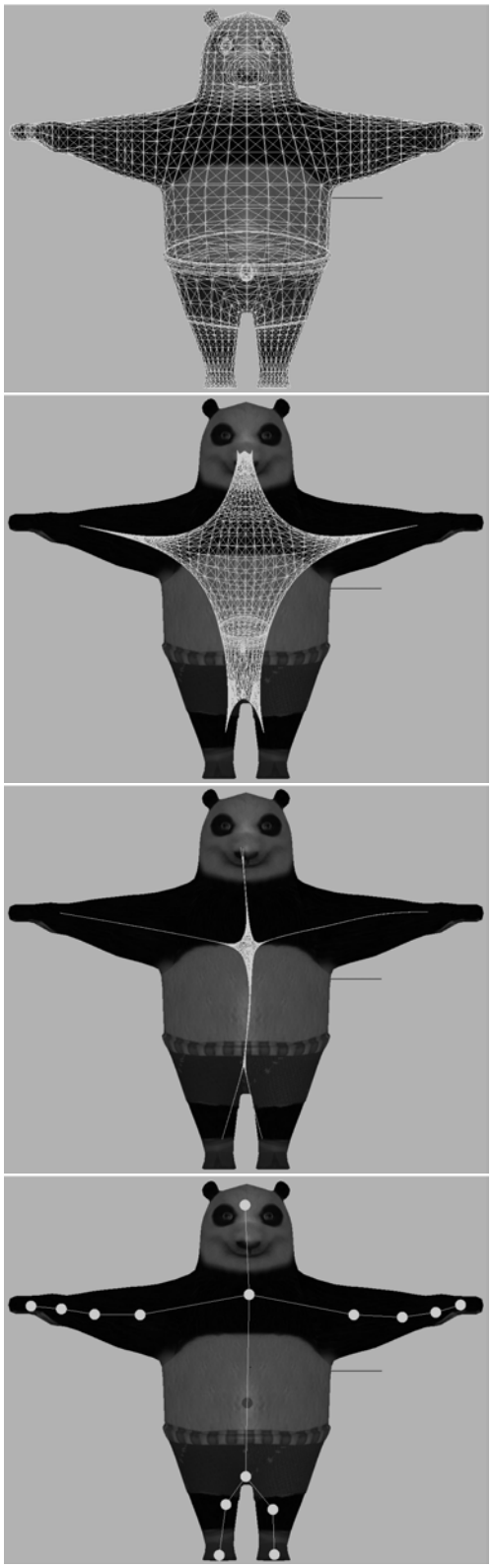


Figure 7: Another example of higher resolution geometry. The extracted skeleton has sparser nodes at the core parts. This feature can be observed, because many faces at core parts are contracted into the same region. The points are also moved towards the mesh boundary of the limbs, because of the shifting of the control points to the center of its local mesh.

ally rigged one. It can be used as a sufficient supplicant for a manually created and rigged skeleton. The major difference can be observed on the both ends of the worm and in the largest bend. Nodes of the skeleton tree were pushed into their centers of local mesh areas and that is why they were pushed inside, away from the mesh boundary.

9 Conclusion

In this paper we propose a framework for extracting a skeleton and skinning data from the geometry mesh using an iterative mesh contraction. The approach begins with converting the geometry into a mesh graph. This graph is iteratively contracted and a greedy algorithm is applied to choose the most important subset of vertices. In the next step, these vertices are converted into a hierarchical skeleton. Important skinning data such as indices and weights which are controlling the influence of the skinning process over vertices are computed as well. When the data is computed, our framework also provides a way to inspect the skeleton, simulate the skinning deformation process with full GPU support and allow the user to change the skeleton branching, if it is needed. After all, geometry and skinning data can be exported into Collada 1.5 *.dae* file and transferred into an external application to create animations.

The extracted skeletons have sparser nodes at the core parts of the model. This feature can be observed, because many faces at core parts are contracted into the same region. Computed skeletons are independent of the size and resolution of the models. The approach is insensitive to noise, but works only for closed mesh models with 2D manifold connectivity.

References

[1] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, 2008.

[2] Grégoire Aujay, Franck Hétroy, Francis Lazarus, and Christine Depraz. Harmonic skeleton for realistic character animation. In *Symposium on Computer Animation*, pages 151–160, 2007.

[3] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 72, 2007.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, pages 558–565. MIT Press and McGraw-Hill, first edition, 1990.

[5] Nicu D. Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007.

[6] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 99*, pages 317–324, 1999.

[7] Tamal K. Dey and Jian Sun. Defining and computing curve-skeletons with medial geodesic function. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 143–152, 2006.

[8] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[9] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 954–961, 2003.

[10] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 479–487, 2005.

[11] Pin-Chou Liu, Fu-Che Wu, Wan-Chun Ma, Rung-Huei Liang, and Ming Ouhyoung. Automatic animation skeleton construction using repulsive force field. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 409, 2003.

[12] Sony Computer Entertainment Inc. *COLLADA Digital Asset Schema Release 1.5.0*, apr 2008.

[13] Marek Teichmann and Seth Teller. Assisted articulation of closed polygonal models. In *SIGGRAPH '98: ACM SIGGRAPH 98 Conference abstracts and applications*, page 254, 1998.

[14] Lawson Wade and Richard E. Parent. Fast, fully-automated generation of control skeletons for use in animation. In *CA '00: Proceedings of the Computer Animation*, page 164, 2000.

[15] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 644–651, 2004.

[16] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 496–503, 2005.

Terrain Rendering with the Combination of Mesh Simplification and Displacement Mapping

Zsolt Fehér*
Supervised by: Zoltán Prohászka†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest / Hungary

Abstract

Today's graphics hardware is not capable of displaying arbitrarily detailed terrains in real–time. Above a certain number of triangles rendering becomes too slow, frame rate drops below 30. Known methods – such as Level of Detail (LOD) or Realtime Optimally-Adapting Meshes (ROAM) – simplify the triangle meshes to maintain speed. With fewer triangles real–time speeds are possible, but it decreases the visual quality. This paper presents a fast, real-time method that combines triangle mesh simplification with pixel shader displacement mapping. This method first builds up an approximate low resolution triangle mesh and applies displacement mapping on that. In the pixel shader it computes the intersection of rays and the terrain. For the best result, it combines linear search with secant search. The result is independent from the resolution of the heightmap and is capable of displaying terrains without decreasing detail.

Keywords: Terrain rendering, Real-time, Displacement mapping, Ray tracing, Linear search, Secant method

1 Introduction

Usually height values of terrains are stored in heightmaps, i.e. two dimensional grayscale textures. Above a certain resolution of the heightmap, too many polygons need to be displayed if we draw triangles between each height point. The result will not be real-time. For example, a heightmap with 2049 x 2049 points would mean more than 8 million triangles. At present, an average GPU could display it only below 30 fps. There are known methods, which simplify the triangle mesh. Most popular are Level of Detail (LOD) and Realtime Optimally-Adapting Meshes (ROAM). As they display fewer triangles, frame rate is higher and acceptable, but visual quality is reduced. It is also possible to display the land with displacement mapping. This could be faster and independent of the resolution of the heightmap, but also could be inaccurate.

*zsolt.feher.88@gmail.com
†prohaszka@iit.bme.hu

This paper presents a method that combines triangle mesh simplification with displacement mapping. It combines the advantages of both methods. With reduced number of triangles, high frame rate is possible. With displacement mapping there is no loss in detail. The triangle mesh helps displacement mapping to be faster and much more accurate. The algorithm first builds up an approximate low resolution triangle mesh above the real relief, then it uses displacement mapping on that. The pixel shader of the GPU determines the ray, and on a segment of the ray, it searches for the intersection with the real terrain. Minimizing inaccuracy, first it uses linear search, and finally refines the result with secant search. The result is fast, independent of the resolution of the heightmap, and there is no degradation in detail.

2 Related Work

Graphics hardware is only able to display a limited number of triangles in real-time. For acceptable frame rate and visual quality several algorithms have been developed in the last decades. These algorithms reduce the number of triangles without significantly decreasing the visual quality. An often used method is LOD [4] [15]. Simplest version divides the surface to quads. In the quad where the camera is, the resolution of terrain is not changed. In distant quads the resolution is highly reduced. The border between two quads is noticeable and could be annoying. Due to the different resolutions, gaps could appear.

Another solution is the ROAM [5]. This algorithm splits triangles into two smaller triangles if the difference between the original and the new triangles is too big, and merges neighboring triangles if the difference is small. This algorithm considers the variety of surfaces. Flat part of the terrain would only consist of few triangles. It also considers what is visible from the camera. Far parts are displayed with fewer triangles than near parts. Both algorithms have an annoying problem. When the resolution of the triangle mesh changes somewhere, this change is instant. The surface pops between two frames, which is easy to notice.

I developed a method that simplifies the triangle mesh.

First the terrain consists of a few big triangles. The algorithm recursively examines every triangle if it is necessary to split it into two equal triangles. A triangle won't be split if the difference between the original and the new one is small. The algorithm handles the variety of surfaces and the triangle mesh doesn't change during running. However gaps could appear on the surface, due to the different size of triangles. Figure 1 shows a simplified terrain with gaps produced by this algorithm. There exist methods to fill these gaps, but they require additional computation and still create difference from the original terrain.

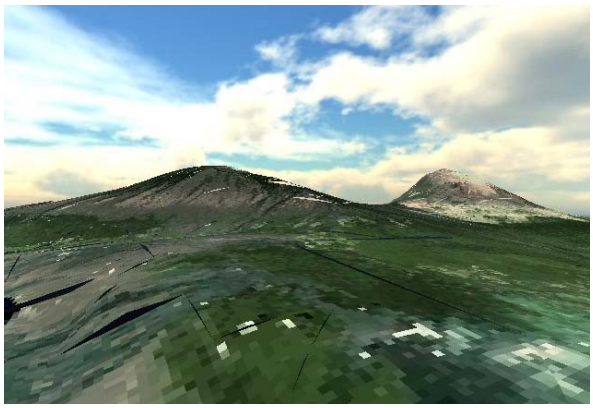


Figure 1: Simplified terrain with gaps

It is also possible to display a terrain without using triangle mesh. Using displacement mapping on a quad is capable of visualizing 3D surfaces. There are several methods that calculate the surface in the pixel shader [1], including parallax mapping, linear – binary search, secant method, cone stepping, sphere tracing etc. Displacement mapping could be really fast, due to the minimal polygon number. Unfortunately it could be also inaccurate.

The widely used per-vertex displacement mapping method (see [1]) differs from our approach, since it uses the vertex shader only, while our method uses both vertex and pixel shaders for displacement mapping.

There also exist a few approaches that combine triangle mesh with displacement mapping [2] [3]. In [2] a similar approach is used for vegetation visualization on orthographic landscape. Our approach is addressed for displacement mapping in general, for example to be used by visualizing reliefs. By our approach, the input is a homogeneous, high resolution height map, while [2] uses different datasets as the input of the visualizing algorithm. In [2], the determination of the height offset is not detailed, in our method it is calculated correctly and based on the height map.

3 Approximate Low Resolution Triangle Mesh

With basic displacement mapping we encountered a serious problem. Displacement mapping is usually used to increase the detail of a surface without using more polygons. Displacement mapping algorithms usually assume that the camera doesn't go below the original surface, and doesn't fly between bumps. In our application we would like to fly among the hills. If we draw the displaced terrain below the polygons, the terrain disappears when camera move below them. A better solution is to create the terrain above the polygons. Then the camera can fly between hills, but for horizontal and above horizon rays the terrain doesn't appear. When a ray doesn't intersect the original polygons, the pixel shader algorithm doesn't start calculating the intersection with terrain (see Figures 2and 3).

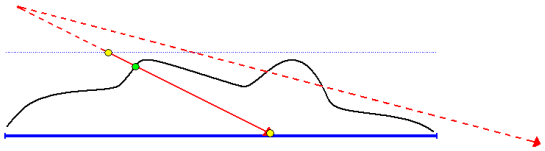


Figure 2: Ray doesn't intersect polygons

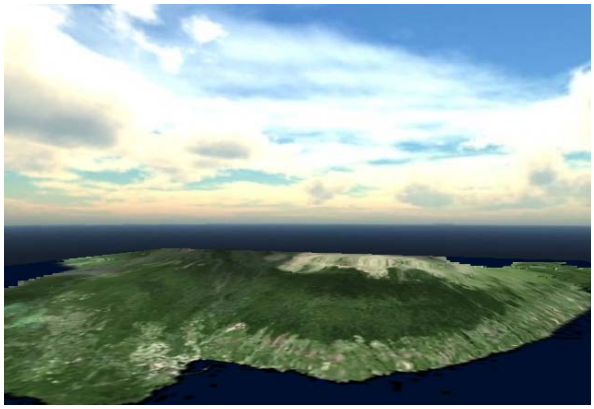


Figure 3: Terrain is not displayed above the base triangles (without using bounding box)

A known solution to this problem is the application of a bounding box. It builds up a box around the displaced surface, therefore every ray would intersect a polygon, and the pixel shader could determine the intersection with the terrain.

In the project we developed a faster method for this problem. It builds up an approximate low resolution triangle mesh above the real terrain in three steps. On this mesh it applies displacement mapping. Since the real terrain is very close to the mesh, the searches in pixel shader will be much faster.

3.1 First Step: Low Resolution Triangle Mesh

First the algorithm creates a low resolution triangle mesh. It uses height values from the height map. It doesn't use all pixels from height map, but skips a certain number of pixels. The resolution of the new mesh can be 16×16, 32×32, 64×64 or 128×128 quads. The optimal choice depends on the graphic card. On an Nvidia Geforce 8400 M the best result was observed at 32×32, but on an 8800 GT we got better outcome with 128×128. For arbitrary resolution, it is important that the resolution of the heightmap is $2^n + 1$.

3.2 Second Step: Push Up the Quads

For each quad[1], the algorithm examines the height values covered by a quad. It computes if a point is above the quad, then stores the highest among them. After the highest point above the quad is determined, the whole quad is being pushed upward to the maximum point. To do this, it is necessary to calculate the distance between the proper triangle's plane and the point. A simple method is to compute intersection between the plane and a vertical line. Place vector $\mathbf{r_0}$ of the line is at 0 height.

$$t = \frac{(\mathbf{p} - \mathbf{r_0}) \cdot \mathbf{n}}{\mathbf{v} \cdot \mathbf{n}} \tag{1}$$

where $\mathbf{p}$ is a point of the plane[2], $\mathbf{r_0}$ is a point of the line, $\mathbf{n}$ is the normal vector of the plane and $\mathbf{v}$ is the direction vector of the line. It is easy to compute the plane's normal vector from the vertices of the triangle.

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}) \tag{2}$$

where $\mathbf{a}$ and $\mathbf{b}$ are vertices of the triangle. Since the point of the line is at 0 height, parameter t is the exact height of the intersection. The difference between t and the real height value shows how far the quad should be raised. The algorithm stores in an array how each quad has been pushed up. This is used in step three.

3.3 Third Step: Fill Gaps

Each quad rose differently, scales of pushes are different at neighbors, therefore gaps appear between quads. These gaps should be removed. For every quad at every vertex, the algorithm examines the other (maximum three) vertices at that point and determines which one is the highest. The vertex (not the entire quad) should be pushed to the height of the highest vertex at that point. After every vertex raised to the proper height, there will be no gaps.

With these three steps, we got a low resolution terrain. The original terrain is never above the new surface but is very close to it.

[1] A quad consists of two triangles.

[2] In this case any vertex of the triangle

4 Displacement Mapping on Approximate Triangle Mesh

If we apply displacement mapping on the approximate triangle mesh, we get better results than using it with bounding box, because the triangle mesh is very near to the real surface. The relevant part of the ray is shorter, and with linear search the intersection could be found mostly within a few steps.

The heightmap texture stores height values between 0 and 255. These will be normalized to the range 0 and 1 on the GPU. In the rest of the article we assume that both $[U\ W\ H]$ coordinates are normalized to 0 and 1.

During basic displacement mapping in the pixel shader we calculate which segment of the ray falls between 0 and 1. On this segment we search for the intersection with the terrain. For this new method, the segment of the ray that enters below the triangle mesh usually starts far below 1. Therefore the segment is shorter, and the terrain is very near to the entry point. An example can be seen in Figure 4.

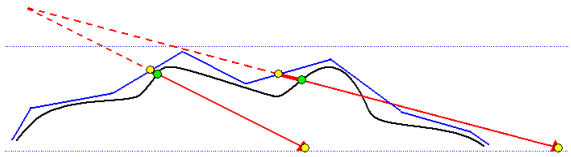


Figure 4: On approximate triangle mesh the segments of the rays are shorter and the terrain is nearer

The binary search or the secant method improved only slightly due to the shorter segment, but linear search became significantly faster. The search begins close to the intersection, generally the intersection is found within a few steps. Using 4 linear search steps, and 1 secant search step, the displayed terrain looks good (Figure 5).

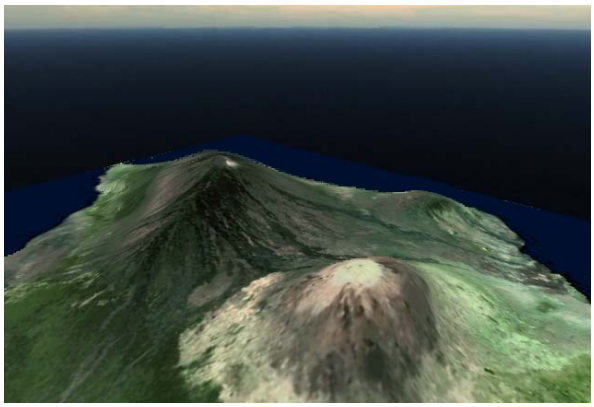


Figure 5: Terrain with 4 linear and 1 secant search steps.

For steep rays, the result is satisfactory, but when the camera is at low height, the upper part of the terrain

doesn't appear correctly. For near horizontal rays, the result is wrong. The triangles above the horizon show upside down, far away repeated terrain. Figure 6 shows the terrain rendered with basic linear search.

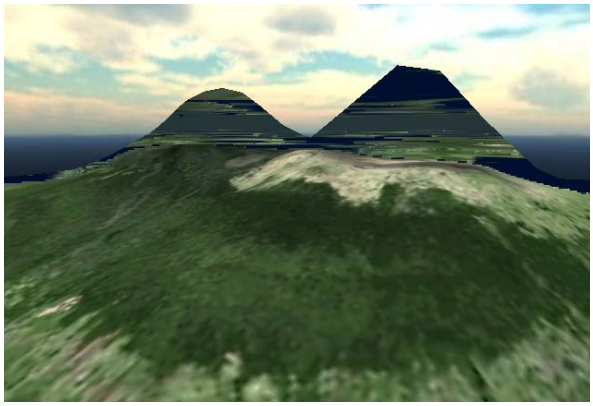


Figure 6: The problems of basic linear search.

4.1 Upward Rays

Basic displacement mapping assumes that the ray enters at $H = 1$ and exits at $H = 0$. The intersection will be found between these points. The ray never reaches $H = 0$ if it starts somewhere between 0 and 1, and is going upwards. The basic algorithm takes steps towards 0. On upward rays, it would also step towards 0. In this case these steps go behind the camera, because 0 can be reached only there. This is the reason for the appearance of the upside down terrain above the horizon. To solve this problem we should handle upward rays differently. It is sufficient to examine the ray's direction vector's H (3^{rd}, vertical) coordinate. It is positive for upward rays. The determination of the ray's endpoints needs to be changed. The endpoint of an upward ray is at $H = 1$ instead of $H = 0$. Results show that the above problem is solved by this change. Upward rays also find intersection (shown in Figure 7), but there are still problems with pixels (rays) close to the horizon.

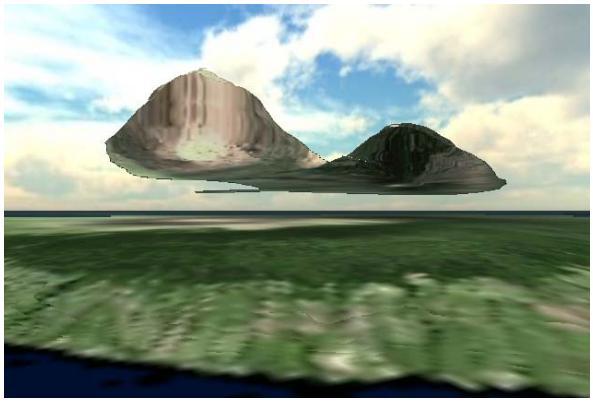


Figure 7: The problem of near horizontal rays.

4.2 Near Horizontal Rays

The original algorithm steps vertically the same ranges between the in and out points for linear search. When a ray is flat, a small vertical step could result in huge step forward on the ray. Figure 8 shows an example. It is possible that one step skips the whole terrain. In this case the intersection cannot be found. This is the reason why a part of the terrain doesn't appear in Figure 7. We found two different solutions for this problem: Exponential Search and Equidistant Linear Steps.

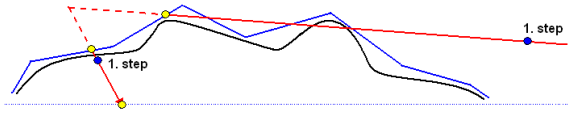


Figure 8: The problem of flat (near horizontal) rays.

4.3 Exponential Search

Standard linear search has the problem that for flat rays it uses too large steps and it steps over the whole terrain. A better solution is that the algorithm steps rather exponentially than linearly. At first the steps are very small, but increase exponentially. However, it reaches the end of the ray with finite steps, similarly to standard linear search. The range of the steps are:

$$H = 1 - 1/2^{N_{iter}-1-i} \tag{3}$$

where N_{iter} is the number of the iterations and i is iteration variable. H is the height of the steps on the ray segment. The steps go from the enter point – the H is 1 there, and go forward to the end of it – where H is 0. The current position is calculated by linear interpolation between the two tips by the following code:

```
uv = uv_in * H + uv_out * (1-H);
```

where `uv_in` the enter point and `uv_out` is the end point. Table 1 shows an example for exponential steps at 10 iterations.

The result is much better than for linear search (shown in Figure 9). Most of the horizontal rays find proper intersections, however not all of them. In a few pixel width strip there is still no intersection. The horizontal and almost horizontal rays are flat, they can have nearly infinite length. On these rays even a very small step skips the whole terrain (see Figure 8).

4.4 Equidistant Linear Steps

The main problem with the standard algorithm is that it steps vertically the same ranges to ensure it always reaches the end of the segment, but it does not consider the ray's

Iteration	Height of step
0	0.998046875
1	0.99609375
2	0.9921875
3	0.984375
4	0.96875
5	0.9375
6	0.875
7	0.75
8	0.5
9	0

Table 1: Exponential Search with 10 iterations

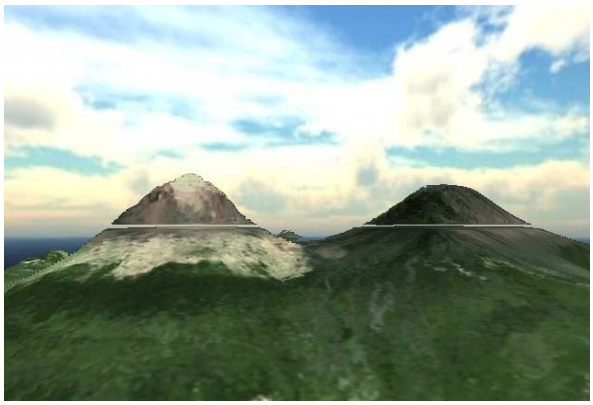


Figure 9: Terrain with Exponential Search, 10 iterations.

length. Therefore the lengths of the steps are not equal for rays of different declination angles.

Instead of stepping vertically, the algorithm should step equal distances. Thus, for every ray the length of the steps would be the same. At flat rays, the algorithm does not skip the terrain. Figure 10 shows the same example, but with the new linear search. It is observable that this time it does not step too much, and the intersection can be found. However a new problem appears. The former algorithms always can reach the end of the segment, but this algorithm can not. For example, if the iteration number is 5, than the upper ray in Figure 10 is unable to reach the real intersection.

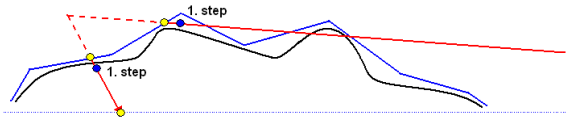


Figure 10: Improved linear search with fix step ranges.

To determine the step's range, at first the length of the ray's segment should be computed. Then a constant is divided by this length. At each step this value is subtracted from H[3].

$$\mathbf{t}_{view} = \frac{\mathbf{M}_{terrrain} * \mathbf{p}_{eye} - \mathbf{p}_{terrain}}{\|\mathbf{M}_{terrrain} * \mathbf{p}_{eye} - \mathbf{p}_{terrain}\|} \tag{4}$$

$$step = c \frac{\mathbf{t}_{viewz}}{(h_{end} - \mathbf{p}_{terrain_z})} \tag{5}$$

where $\mathbf{M}_{terrrain}$ is the UWH matrix, $\mathbf{p}_{eye}$ is the position of the camera and $\mathbf{p}_{terrain}$ is the UWH position of the pixel. h_{end} is 0 when the ray goes downward, 1 otherwise.

The constant depends on the resolution of the triangle mesh. If the resolution is high, then the real terrain is nearer, thus smaller steps give better results, so the constant is smaller. At lower resolution the distance between the triangle mesh and the real terrain could be larger, thus greater constant is better. E. g. at 32×32 resolution a good constant was 0.025, and 5 steps were enough.

For a small part of the pixels, this linear search could not reach the real terrain, thus the intersection cannot be found. Mostly this problem occurs for those rays which enter below the triangle mesh, but they are above the real terrain. To solve this problem, the algorithm handles these rays separately. If the intersection was not found, the search continues with more iteration. In the new iterations the steps could be large, e.g. twice as big as originally. As these rays are in minority, the algorithm would not be significantly slower. If the increased number of steps still cannot find an intersection (it is possible that they never would), then neither the pixel will be colored, nor the Z-buffer will be written.

Using standard linear search, the terrain will be striped. Therefore, the search continues after an intersection is found, but using secant search. The two start points of the secant search are the last two points of the linear search. With the penultimate linear step, the algorithm determines a point where the terrain is still below the ray. The last step shows that the terrain is above the last point. The surface of the real terrain intersects the ray between these two points. The secant method finds an accurate intersection quickly, thus the strips are eliminated.

As Equidistant Linear Steps perform much better than the Exponential Search, the Exponential Search is not used in the final version.

5 Results

It is hard to determine the optimal constant values in the algorithm such us triangle mesh resolution, first linear search's iteration number, second linear search's iteration number, step ranges, secant search's iteration number etc. The performance of the described algorithm is highly affected by the length of the ray sections which fall between the course triangle mesh and the real terrain. Average of these lengths depend on the local roughness and local curvature of the real terrain, but is nearly independent

[3] At beginning H is 1.

of the local steepness.Rendering speed also highly depend on graphics hardware. The algorithm was developed and tested on an Nvidia® Geforce® 8400M[4]. The viewport resolution was 640×480 pixels. The frame rate dropped at 64×64 quads, thus 32×32 quads was a better choice. With 5 linear steps most of the intersection was found. For pixels, where the search could not reach the intersection, 10 further linear steps followed. If the search was still inefficient, the pixel became transparent, thus other part of the terrain or the skybox could appear. However if the linear search entered below the real terrain, 2 secant search steps refined the result. Figure 11 shows the terrain with these values. The frame rate was between 55-100 fps[5].

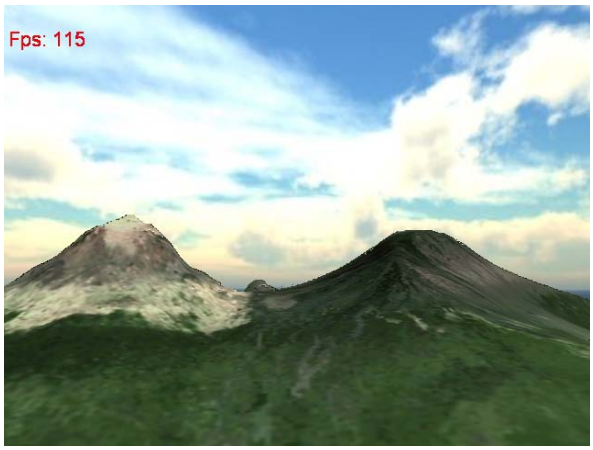


Figure 11: Final terrain.

As seen in Figure 11 the terrain appeared correctly. The algorithm finds the proper intersection for almost every pixel. However, at some pixels the algorithm makes mistakes. These mistaken pixels mostly appeared at the edge of hills.

Using the same paremeters, the frame rate on a desktop PC with Nvidia® Geforce® 8800 GT was much higher, over thousand fps. After changing parameter values, the algorithm was more accurate, pixel errors barely appeared. The triangle mesh resolution was set to 256×256, step ranges was the quarter of original, linear step numbers doubled. The result was still over hundreds of fps and with unnoticeable errors. Figure 12 shows an example, where the heightmap's and the texture's resolution was 2049×2049. Table 2 shows results on different GPUs and settings. As the rendering time was highly dependent of camera position, four points of view has been selected. It was intentional to use fundamentally different views (see Figure 13). Frame rate tests for different configurations were tested on these fixed views. Refresh times show minima and maxima of the four measured values.

[4]Notebook version

[5]Depends on percentage of terrain on screen

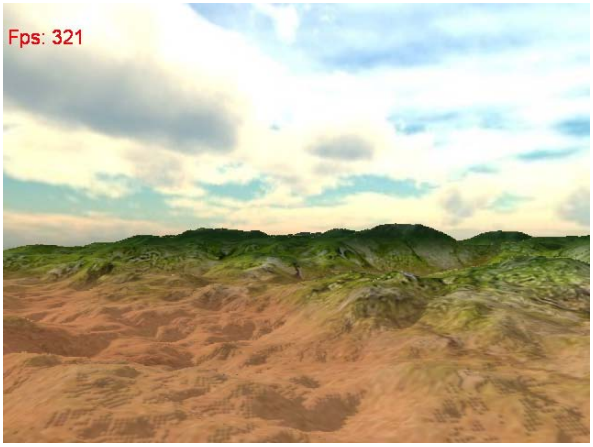


Figure 12: Final, 2049×2049 resolution terrain on 8800 GT with 256×256 quads.

GPU	Heightmap	Mesh	Speed[ms]
8400 M	257×257	32×32	9.52–18.87
8400 M	2049×2049	32×32	9.62–20
8400 M	2049×2049	256×256	22.22–41.67
8800 GT	257×257	32×32	0.55–0.8
8800 GT	2049×2049	32×32	0.55–0.83
8800 GT	2049×2049	256×256	3.7–3.82

Table 2: Rendering times

6 Conclusion and Future Work

There are several known algorithms that use only the pixel shader or only the vertex shader for displaying terrains. We developed a new technique that combines the pixel shader and vertex shader techniques. It resulted in a faster and more accurate algorithm than widely used ones. The main advantage of this new method is that it is independent of the heightmap's resolution. The approximate triangle mesh's resolution is constant, does not consider the heightmap's resolution and the displacement mapping is also independent of the heightmap's size. The frame rate is similar e.g. at 257×257 and at 2049×2049 resolution heightmaps.

However it is hard to determine the balance between speed and accuracy. If the decision is to be more accurate, the frame rate decreases. The method is also capable of reaching high frame rate on slower GPUs, but then accuracy has to be decreased.

In the future we shall make the algorithm better. E.g. improve the displacement mapping searches or make the triangle mesh dynamically changeable. A new method could be also promising: using height mipmaps the algorithm would not make mistakes, the proper intersection always could be found rapidly. Our research is going on by utilizing maps for local height minima and maxima. Instead of [2], the resolutions of these extremum maps are decreased in our research, similarly to the Mip-Map ap-

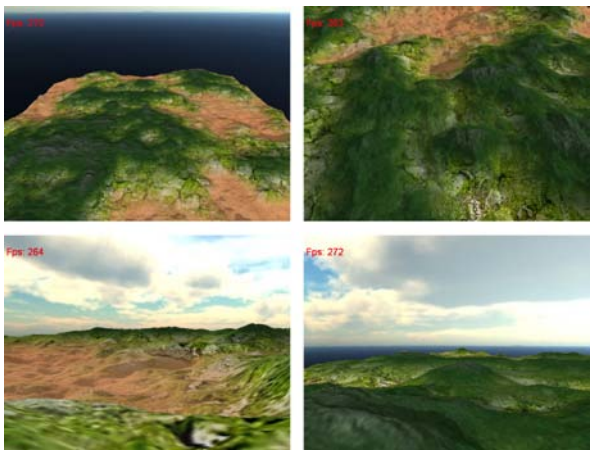


Figure 13: These four viewpoints were selected to be used during frame rate test of the algorithm on different platforms with different options. Shown frame rates were achieved on Geforce 8800 GT, 2049×2049 height map, 256×256 quads.

proach.

After a fast, accurate, detailed terrain displaying algorithm is developed, it could be improved by adding new elements, such as new detailed textures close to camera, dynamic light and shadows, vegetation, water etc.

Acknowledgement

This work has been supported by the Teratomo project of the National Office for Research and Technology, and OTKA K-719922 (Hungary).

References

[1] László Szirmay-Kalos, Tamás Umenhoffer. *Displacement Mapping on the GPU – State of the Art*. Computer Graphics Forum, 2008.

[2] Stephan Mantler, Stefan Jeschke. *Interactive landscape visualization using GPU ray casting*. Graphite, 2006.

[3] Christian Dick, Jens Krüger, Rüdiger Westermann. *GPU Ray-Casting for Scalable Terrain Rendering*. Eurographics, 2009.

[4] Renato Pajarola, Enrico Gobbetti. *Survey of semi–regular multiresolution models for interactive terrain rendering*. The Visual Computer 8: International Journay of Computer Graphics, pp. 583–605, 2007.

[5] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, Mark B. Mineev-Weinstein. *ROAMing Terrain: Real-time Optimally Adapting Meshes*. IEEE Visualization, 1997.

[6] Jonathan Dummer. *Cone Step Mapping: An Iterative Ray–Heightfield Intersection Algorithm*. http://www.lonesock.net/files/ConeStepMapping.pdf, 2006.

[7] Barry Minor, Gordon Fossum, Van To. *Cell Broadband Engine Optimized Real–time Ray–caster*. 2005.

[8] Brian Smits, Peter Shirley, Michael M. Stark. *Direct Ray Tracing of Displacement Mapped Triangles*. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, 2000.

[9] Arul Asirvatham, Hugues Hoppe. *Terrain Rendering Using GPU-Based Geometry Clipmaps*. GPU Gems 2, pp. 27–46, 2005.

[10] Alex A. Pomeranz. *ROAM Using Surface Triangle Clusters (RUSTiC)*. Department of Computer Science, University of California, 2000.

[11] Jonathan Blow. *Terrain Rendering Research for Games*. SIGGRAPH 2000 Course 39, 2000.

[12] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, Gregory A. Turner. *Real–Time, Continuous Level of Detail Rendering of Height Fields*. Proceedings of SIGGRAPH 96, pp. 109-118, 1996.

[13] Stefan Rottger, Wolfgang Heidrich, Philipp Slusallek, Hans-Peter Seidel. *Real–Time Generation of Continuous Levels of Detail for Height Fields*. WSCG Proceedings 98, pp. 315-322, 1998.

[14] Jens Schneider, Rudiger Westermann. *GPU–Friendly High–Quality Terrain Rendering*. Journal of WSCG, 2006.

[15] Wikipedia. *Level of Detail*.

GPU-supported bubble and foam rendering

Tamás Huszár *
Supervised by: László Szécsi†

Department of Control Engineering and Information Technology,
Budapest University of Technology and Economics,
Budapest, Hungary

Abstract

Several types of foam can be found both in nature and artificial environments; yet it is rare in computer graphics due to its complexity. Modelling foam structure and dynamics by simulating the underlying bubble structure has a high computational cost. To model such a complex phenomena we need to use serious simplifications while maintaining realism and detail.
In this paper we propose a method for rendering dense soap foam in real time. We first build a foam blob — from realistic soap bubbles — which has a solid inner structure. We use a hybrid method based on ray tracing and 2D billboards to render dense foam constructed from hundreds of these blobs. To model foam behaviour and interaction, we present a simple particle based physics simulation approach. While our method is capable of rendering foam featuring a large number of bubbles, it has certain limitations we also discuss in this paper.

Keywords: Bubble rendering, foam rendering, impostors

1 Introduction

Presenting natural phenomena like smoke, fire or fluids in a realistic way is a tough challenge in computer graphics. Even though the equations describing the physics of these phenomena are known, the exact calculations are too complex to perform in real time. Today's graphics hardware requires some intuitive simplifications or artistic input to efficiently present these phenomena in real-time applications like computer games.

Bubble and foam simulation falls into the above category. The structure of dense foam built of soap bubbles exhibits large complexity. While modelling a physically correct soap bubble is an easy task, building complex foam structures from individual bubbles cannot be done in real time on current hardware.

In this paper, we propose a method to render and simulate dense soap bubble foam. We give an intuitive simplification of foam structure which enables us to simulate realistic foam with the speed, detail and quality necessary for real-time applications. After the introduction and previous work, we give a short overview of bubble simulation, discussing the actual techniques used in our method in section 3, including our solution for efficiently modeling multiple connected bubbles. Section 4 introduces the idea of building foam from blobs of soap bubbles. We discuss the structure of these blobs and provide two different methods of storing and rendering blob structure. In section 5, the technique of building actual foam of the blobs is discussed, including a basic physics simulation described in section 6. In the next section we present our result, and provide possible enhancements and future work in further sections, including the conclusion of this paper.

2 Previous Work

Interference phenomena required to understand bubble physics were described by Dias [1]. Later, Glassner gave a thorough overview on several aspects of soap bubble physics, including soap film interference and geometric structure of multiple soap bubbles [3, 4]. Most attempts to model bubble and foam structures are offline methods based on ray tracing [7], and even these offline approaches [6] use simplified reflection model to render the inner dense parts of the foam to maintain reasonable rendering times.

Recent articles present real-time approaches and use the GPU to simulate bubble formations. Sunkel simulates a magnitude of hundreds soap bubbles in real time using simplified reflection and lighting model [8].

3 Realistic rendering of soap bubbles

3.1 Soap film interference

In order to simulate foam, we must first understand the physics of soap bubbles, and provide an efficient way to render soap films and bubble structures. Basically soap bubbles are gas trapped in a thin fluid layer. Because of surface tension and the inside pressure of the contained gas, the surface of a bubble tends to be minimal. This means soap bubbles can be easily modelled as spheres;

*hthomas92@gmail.com
†szecsi@iit.bme.hu

this is a common simplification used by most approaches. The interference phenomenon on bubble surfaces can be understood by examining soap film reflection — light interference caused by reflection on two parallel surfaces. Usual soap films are 1–2000 nm wide and have a refraction index of 1.4. Given these values, the intensity change of the reflected light can be calculated by the following equations [4, 5].

$$ps = \frac{4\pi}{\lambda} nd \cos\vartheta_i$$

$$R_f = 1 - \cos\vartheta_i$$

$$I_r = I_i 4 R_f \sin^2 ps$$

The intensity change I_r depends on the incoming intensity I_i, the reflection factor R_f and the phase shift ps. The phase shift can be calculated using the wavelength λ and the incident angle of the light ϑ_i, the index of refraction n and the film width d. The refraction index is calculated using a Fresnel approximation, the film width and the refraction index are constant (we can perturb the film width with random noise to make the bubble more realistic, simulating film thickness changes caused by air pressure variation). By using these simplifications, the intensity change is only dependent on the incident angle and the wavelength, so it can be easily computed or stored in a texture. We used the representative wavelengths of the RGB components, as the achieved quality is acceptable and it is more efficient than calculating the values over a continuous spectrum. Using these equations and simplifications, a soap film shader can be easily constructed.

3.2 Soap bubble geometry

As we saw earlier, a single soap bubble can be approximated by a sphere. However, for modelling bubble structures, we must compute the shared wall film between the bubbles. Based on Glassner's observations, three soap films always meet at 120° angle and the mutual wall is spherical itself [4]. First, considering two intersecting bubbles, we must determine this auxiliary sphere's centre and radius. The easiest approximation would be a simple planar soap film between the two bubbles. This is an acceptable approximation for distant bubble formations but unrealistic when examined closely. In his article, Glassner gives a formula, in which he exploits the aforementioned 120° property. In a general situation without proper physical simulation, when bubbles are spheres of random radii, this rule does not hold. To overcome this we provide a simple but intuitive and visually convincing approximation.

Figure 1 shows the geometry in a 2D slice. We used a simple observation: the tangent T_c is the angle bisector of the angle determined by the intersection of the spheres and the centres ($AMC\angle$). If we extend the bubble model by keeping this rule, but omitting the 120° restriction, we still get acceptable results with reasonable calculation complexity.

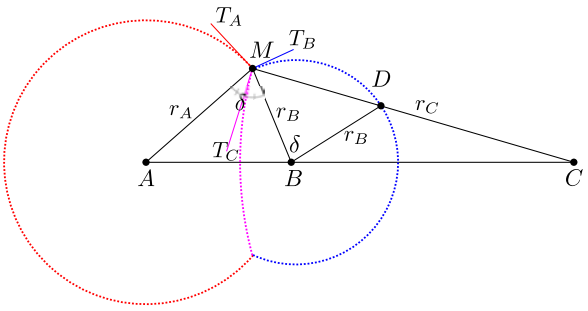


Figure 1: Geometry of bubble walls. Point C is the centre, r_c is the radius of the sphere which forms the common wall of sphere A and B.

First, using the observation above, we realize that the triangles AMC and BDC are similar. This means that $\overline{DC} = r_c r_B / r_A$. As $\overline{MC} = \overline{MD} + \overline{DC}$, we get the following equation (assuming $r_A > r_B$):

$$r_c = \frac{r_A \overline{MD}}{r_A - r_B}$$

The length of MD can be calculated using the cosine rule two times in triangles AMB and MBD.

$$\overline{AB}^2 = r_A^2 + r_B^2 - 2 r_A r_B cos\delta$$

$$\overline{MD} = \sqrt{r_B + r_B - 2 r_B^2 \cos\delta} = r_B \sqrt{2 - 2\frac{r_A^2 + r_B^2 + \overline{AB}^2}{2 r_A r_B}}$$

Finally, we can calculate the centre using the law of cosines the third time, in triangle AMC.

$$\overline{AC} = \sqrt{r_A^2 + r_C^2 - 2 r_A r_C \cos\frac{d+\pi}{2}}$$

4 Modeling foam using soap bubbles

4.1 Foam structure

While soap foam consists of several soap bubbles, the inner structure of the foam is so complex, that simply modelling it as individual bubbles is not a working solution. Today's real-time methods are capable of rendering hundreds of bubbles, but this is far from the complexity of dense foam. To overcome this, we examined the macrostructure of soap foam. On a large enough scale, below the surface bubbles of dense foams, we see a nearly diffuse and opaque whitish body, hiding the deeper microstructure of the foam. Our idea was to model a foam blob possessing this property. The surface of this blob is built of realistic soap bubbles, and the inside is approximated by an artist-drawn bubble texture representing the inner structure of the foam. This creates the impression of a dense interior with individual transparent bubbles protruding from the blob.

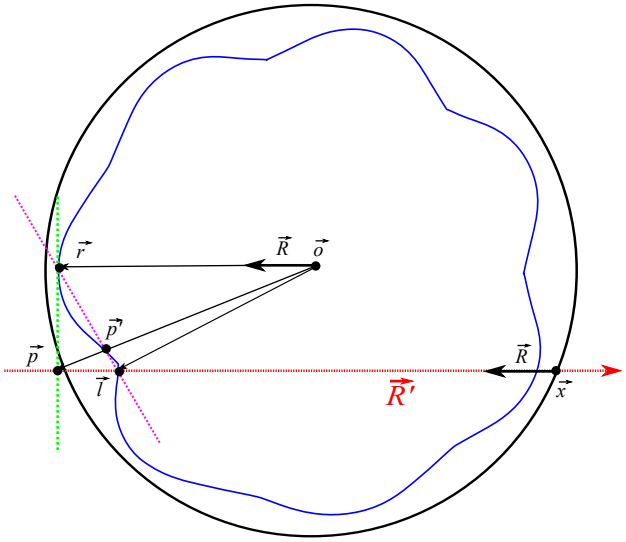


Figure 2: Using the distance impostor technique to model foam blobs. The curve inside the bounding sphere visualizes the distance values stored in the cube map. $\overrightarrow{R'}$ is the view ray and $\vec{l}$ is the intersection point provided by the algorithm.

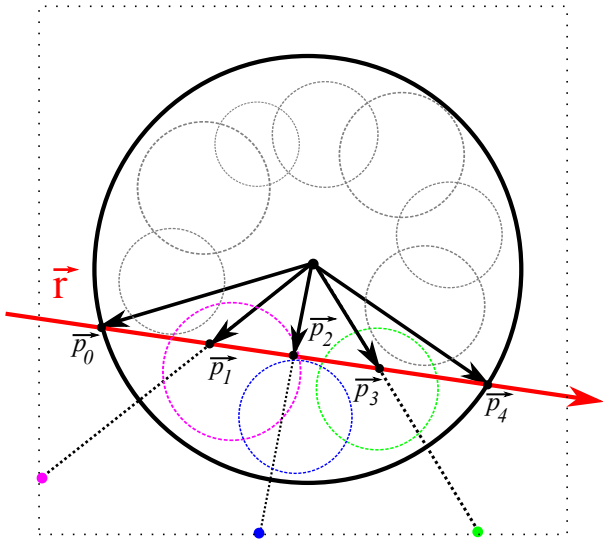


Figure 3: Illustration of the bubble ID technique. The view ray $\vec{r}$ is sampled in points $\vec{P}_0$ - $\vec{P}_4$. The dotted lines show the bubble id stored in the cube map in the designated direction.

The outer bubbles are spheres drawn using the soap film shader, including the walls between neighbouring bubbles. Due to the complex nature of the blobs, classic polygon-based rendering is not a feasible solution. On the other hand, classic ray tracing is too slow, as complex foam can contain thousands of bubbles. We needed a data structure to store bubble data that makes fast and efficient intersection of the blob and view rays possible.

4.2 Blob intersection using a distance impostor

Our first proposed method is based on the distance impostor technique [9]. The original technique is used for calculating position dependent reflections using environment maps. It stores environment geometry in a cube map, and uses it to iteratively calculate surface points intersected by reflection rays. We imagine the foam blob as an entity trapped in a cube, represented by a cube map storing the distance between the blob surface and the centre in every texel. Now finding the intersection of the ray and the blob is the same problem as the one stated above, with one minor difference: our rays come from outside of the cube map, not from the inside.

As seen in Figure 2, R' is the original ray. We reverse the direction of this ray to get $\vec{R}$, and place its origin $\vec{x}$ at the second intersection of the original ray and the blob's bounding sphere. As the new ray points towards the near side of the blob (closer to the origin of R'), the iterative search algorithm will find an intersection $\vec{l}$ on this side. Reading a distance value from the side closer to $\vec{x}$ would give a wrong intersection point. Therefore, running the original algorithm with the reversed ray will give the right result.

The required cube map resolution and iteration count depend on the blob geometry. For a common blob consisting of 64 bubbles, which we used for testing, a resolution of 64×64 and a maximal iteration count of 20 were adequate. When using smaller values, visible artifacts appeared near the intersection of the bubbles.

The cube map itself is generated by rendering the bubbles from the centre of the blob. Using a geometry shader and a render target array, only one rendering pass is required. We render a full screen quad, and calculate the world coordinates of the quad's vertices at the near clipping plane for each render target, using different view matrices (a total 6 cameras covering both directions in all 3 axes). In the fragment shader, we compute the farthest intersection point with each of the bubbles, getting the outside surface geometry of the blob. While the intersection calculation is done by checking all the bubbles in the blob, this is not a performance bottleneck as the cube map has to be generated only once during the program startup.

The disadvantage of the method is that we lose the inner structure of the blob as only the outer shell is stored in the texture. As a side effect, this allows the shape of the outer bubbles to be other than spheres. However, when rendering soap bubbles, this does not grant us an advantage, but makes further calculations — like intersection with the inter-bubble wall — impossible. This recognition motivated our second method. Instead of storing distances of the surface, we store the original bubble data, but we heavily reduce the number of necessary intersection calculations.

4.3 Blob intersection by storing bubble identifiers

Once again, we use a cube map to store blob geometry. But instead of distances, we store an ID of the bubble visible from the outside in the given direction. We also have to store the bubble radii and centres in a separate texture or buffer.

First we calculate the intersection of the ray and the bounding sphere, and then we have to find the bubble it intersects first. This would allow us to use an iterative search similar to the one used in the distance impostor technique, but we found that a simple linear search is adequate. This is because finding any texel that contains the ID of the first intersected sphere is sufficient to get an accurate result. As seen in Figure 3, we divide the section of the ray inside the sphere to a fixed number of segments, and then start reading the values from the cube map in the given directions and calculate the intersection of the ray and the corresponding bubble. The first intersection is the one we are looking for.

The advantage of this method is that in most cases, especially if the bubbles are nearly the same size, we will get the result in the first few iterations. Usually the loop ends in the first iteration when the incident angle is high (the ray goes through the middle of the sphere) and the required iteration count increases as the ray gets further from the centre. Also nothing guarantees that we find the right intersection; in theory we can easily skip the right bubble during the linear search. However experiments showed that when using reasonably sized and evenly distributed bubbles, the results are acceptable. A 128×128 cube texture with 10 iterations produced minor artifacts comparable to the distance impostor technique, and it was also slightly faster.

The actual implementation including the cube map generation is similar to the distance impostor technique. The identifiers are stored in the cube map using the same rendering technique, but instead of the distance, the bubble's id is stored. The bubble identifiers and the correspondent radius and centre values are stored in a buffer located in the video memory.

Storing bubble IDs has another advantage over the distance impostor method: not only the first intersection, but also the intersection with interior walls between bubbles can be computed. When using distance impostors, we only preserve surface geometry, which makes the representation of the precise sub-surface structure impossible. When using the bubble ID technique, we have the exact bubble geometry stored in a buffer. We can use this geometry to calculate inner bubble walls. The linear search algorithm will yield a list of bubble IDs along the ray, if we do not stop it after finding the first intersection. Consecutive bubbles in this list are most likely to form a mutual wall, which can be computed as described in Section 3.2 and intersected with the ray. This is also an approximate solution, as internal bubbles not stored in the ID map are not considered and small bubbles can be skipped by the linear search algorithm. However, with similarly sized bubbles of a soap foam blob this rarely happens, and it does not influence visual quality.

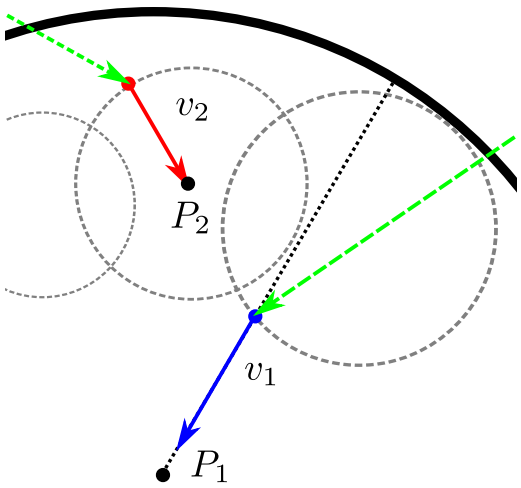


Figure 4: Illustration of the blob normals. The green arrows are light rays. Vector v_1 and v_2 are sampling directions using different calculation methods presented in the paper.

Figure 5: Different foam textures. The texture on the left was used in the final renderings.

4.4 Inner structure and other details

Using any of the techniques presented above, the rendering of the blob is quite straightforward using ray tracing. We calculate the intersection of the blob and the ray coming from the camera. Then we compute the incident angle of the ray and the surface normal in the previously calculated intersection point. We use these values and an environment map to calculate bubble reflections using the soap film shader. We can also calculate the internal walls for neighbouring bubbles.

Finally we have to draw the inner structure using the bubble texture. It can be an artist-drawn image of small bubbles (5), representing the inner structure of the foam, or a computer generated foam image using a complex non real-time simulation. The foam texture can be stored in another cube map. The sampling direction can be adjusted several ways as it is depending on the given blob and foam type. We used a weighted sum of two vectors (4). The first vector (v_1) is the direction of the second intersection

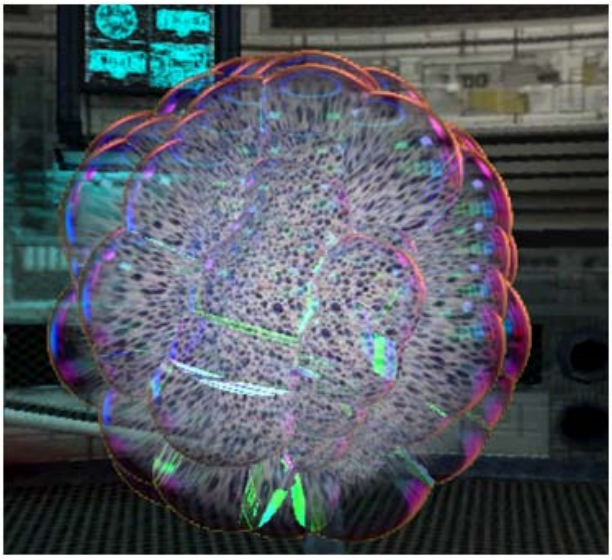


Figure 6: A single foam blob rendered using the bubble id method

of the first intersected bubble (P_1) relative to the centre of the blob (P_1). This spherically projects the texture onto the interior surface of the blob, which is what we get if we remove the outer bubbles. The second vector is the inverse normal of the bubble (v_2) at the first intersection (P_2). Combining these two vectors slightly distorts the original mapping based on the outer bubble geometry.

As stated before, the middle of the blob is opaque, while the outer bubbles are nearly transparent. To achieve this effect, the transparency of the inner texture is set according to the second intersection point's distance from the inner and outer bounding sphere. In the middle of the blob, the second intersection of the current bubble is closer to the centre as the ray is almost perpendicular to the blob's bounding sphere. Near the outer region, the intersection point is farther from the centre, so the blob will be more transparent there.

The last issue is the surface normal of the blob used for lighting equations. The nearly diffuse inner surface should slightly follow the wrapping surface of the outer bubbles. Therefore we used a weighted average of the bubble's normal and the blob's bounding sphere's normal. The normal and the other aforementioned parameters should be fine-tuned and set according to the actual blob structure and the desired foam type. A typical foam blob used in our renderings can be seen on Figure 6.

5 Rendering foam using foam blobs

Realistic dense foam needs to be constructed from many blobs, so we need a fast technique to render them. While ray tracing the blobs could be straightforward, it would be too slow for large foam. We propose a method based on particle systems, that is capable of rendering hundreds of

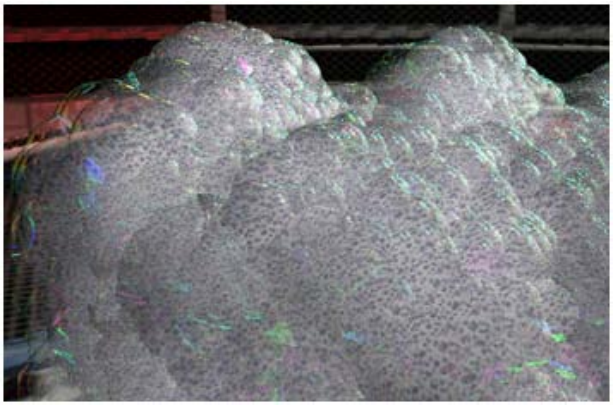


Figure 7: Dense soap foam rendered using the proposed technique

blobs real time. Blobs are rendered as 2D billboards. The ray from the eye position is calculated for all pixels of the billboard, and it is used to render the corresponding blob as described in the previous section. This means we do not have to do the intersection calculations for all blobs, but also means we cannot calculate inter-blob reflections (which would be too slow to use in real time anyway).

The data associated with the blobs are stored on the graphics card in a vertex buffer. The billboards are generated by the geometry shader using this data. The vertex positions in world space are also calculated in the shader, and used in the pixel shader to get the view rays for every pixel. Besides the colour, the depth of the blob is also computed for every pixel to address problems of overlapping particles.

Since the blobs are transparent, we need to sort the particles according depth, in order to use alpha-blending. However, depth is different in the pixels of the billboards, so we have to do the sorting at pixel level. Depth peeling [2] is a technique rendering translucent objects. It basically accomplishes pixel level sorting by using multiple rendering passes to store multiple depth levels for every pixel. We use two buffers to store depth data. First we render the scene depth into the first buffer, and then render it again into the second buffer, but only those pixels that are further than the stored depth in the first buffer. Then we flip the two buffers and repeat. We store the blob identifiers in a third buffer, in a different colour channel for each iteration. In the end, the n closest bubble identifiers will be in the final buffer. In the final rendering pass we render the blobs in order with proper transparency. While this method uses multiple rendering passes and it is generally slow, it provides a real-time alternative to ray tracing.

This method has one serious shortcoming in cases where multiple blobs overlap each other. Let's assume that we use three rendering passes (and store 3 layers of blob depths). Now imagine that the first three blobs are rather transparent, but there is a fourth, solid blob behind them. In this scenario, the resulting foam would be transparent,

resulting a transparent hole in the foam. To overcome this limitation, we use an extra rendering pass to calculate the maximal opaqueness for all visible blobs (not just the 3 closest to the viewer). When we draw the diffuse foam texture, we use this value to plug the unwanted holes in the foam. The main disadvantage of this method is the resource consumption, as we have to calculate intersection points for all blobs (however we do not calculate surface interference, reflections or wall geometry for these blobs). Figure 7 shows dense foam rendered using this technique. All the free parameters were set to grant visually appealing result.

It was not stated explicitly before, but blob rendering techniques require the blob structure to be static. The blob texture is prerendered once and then used for all blobs. This means that all the blobs are the same (it is possible to use several blob textures to construct a fixed number of different blobs). To counteract this limitation, we used transformations to change the size and orientation of individual blobs. This can be easily done real time in the pixel shader during the intersection calculations and grants us more diverse foam. To store these transformations, only one additional float vector is required in the vertex buffer. We can represent the orientation as a quaternion and store it in a four-dimensional vector. The blob size can be the fourth, previously unused coordinate of the position vector. We can also use a transformation matrix to store more general affine transformations, but in our implementation it was unnecessary to do so.

6 Foam physics

To provide even the most basic physical simulation, we must render solid objects beside the foam. As we used environment mapping for reflections and refractions, solid objects must be rendered using a blending technique. We first render these objects, and then blend the foam over them without clearing the depth buffer. This is efficient but this does not handle reflections of solid object on bubbles, or the rendering of transparent objects like smoke or glass.

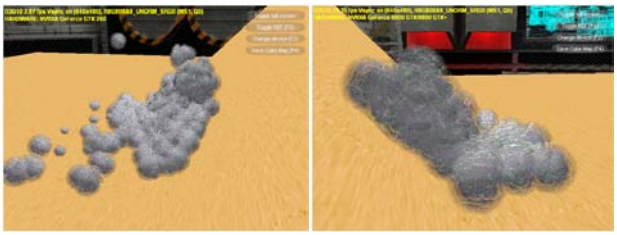


Figure 8: Foam formations sliding down on a slope

To simulate foam dynamics and present the characteristics of this rendering technique we created a basic but fast physical simulation based on particle dynamics. The simulation is able to handle inter-particle forces and outside objects. In the technical demonstration we presented a foam mass sliding down on a slope (see Figure 8). Each blob has a position, mass, and velocity, with forces acting between blobs and other objects. The blobs are represented by their bounding spheres. If two blobs get too close, two types of force can affect them: if they are far enough an attractive elastic force arises, modelling the different parts of the foam sticking together. However, if two blobs get too close, they collide, resisting collapse and giving the foam a solid structure. By properly adjusting these forces and the gravity, a good approximation can be achieved for the desired foam type. The simulation is done on the CPU. Further exploration in this topic could yield more realistic results and boost performance by implementing a physical simulation on the graphics card.

7 Results

We implemented the technique using DirectX 10 and Shader Model 4.0 on an NVIDIA Geforce GTX 260 graphics card. We achieved real-time simulation (32 FPS) of 100 blobs and a total number of 22700 separate bubbles, using the bubble ID technique for calculating intersections. The images were rendered at the resolution of 640×480. The various parameters like iteration count, texture size, and the number of bubbles in a blob were set to imitate soap foam to the highest possible fidelity without visible graphical glitches. Further tweaking these parameters could result in performance increase, while maintaining acceptable graphic quality.

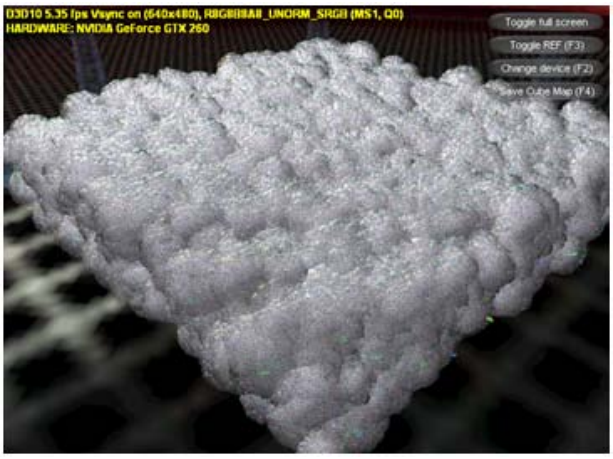


Figure 9: A box shaped form consisting of 5000 blobs

Using these same parameters, simulating between 100 and 1000 blobs the FPS stays above a reasonable rate (around 10). Figure 9 shows a foam formation of 5000 blobs and a total number of 1 135 000 bubbles, rendered at 5 FPS. Given these numbers, in the near future with further optimizations the real-time simulation of foam consisting of hundreds of thousands of bubbles could be possible.

8 Future work

The most serious limitation of the proposed technique derives from the particle based approach. Our blobs are static entities with fixed size and structure, and when we start to divide the foam into smaller pieces comparable to the blob size, it leads to artificial and unnatural results. To overcome this, we propose a possible direction. First, we need a model to adjust blob size dynamically, based on some physical observations, but not too complex to undermine the performance. Another possible way is to locally increase and decrease the simulation resolution (the blob size in this case) by the local foam characteristics and viewer distance. Blobs inside a dense foam or far from the viewer could be merged together, or their simulation and render quality should be otherwise decreased, while larger blobs broken out of the foam should break up into smaller blobs.

9 Conclusion

Bubbles and foam are extremely complex natural phenomena, the formation, motion and optics of which obey complex physical laws practically impossible to simulate in real time. A visually convincing result, however, is feasible with decent data structures and subtle approximations. As with a wide range of natural geometries, the concept of impostors is very helpful. We have shown how a generic impostor technique – the distance impostors – can be modified to represent bubble clusters, and we also proposed a specialized representation that exploits the fact that bubbles are spherical, and allows not only for a more accurate representation, but also for an approximation of internal foam walls. Furthermore, we described algorithms for the simulation and rendering of massive foam composed of the bubble clusters, based on particle systems and the billboard visualization technique. Our method is capable of real-time rendering dense foam consisting of ten thousands of bubbles on modern graphics hardware.

Acknowledgement

This work has been supported by the Teratomo project of the National Office for Research and Technology, and OTKA K-719922 (Hungary).

References

[1] L.M. Dias. Ray tracing interference color. *IEEE Computer Graphics and Applications*, 11(2):54–60, 1991.

[2] C. Everitt. Interactive order-independent transparency. *White paper, nVIDIA*, 2(6):7, 2001.

[3] A. Glassner. Soap bubbles: Part 1. *IEEE Computer Graphics and Applications*, 20(5):76–84, 2000.

[4] A. Glassner. Soap bubbles: Part 2. *IEEE Computer Graphics and Applications*, 20(6):99–109, 2000.

[5] K. Iwasaki, K. Matsuzawa, and T. Nishita. Real-time rendering of soap bubbles taking into account light interference. In *Computer Graphics International*, pages 344–348, 2004.

[6] S. Rosenbaum and M. Bergbom. Foam. http://cs.stanford.edu/people/rosenbas/foam, 2007.

[7] Y. Sun, F. D. Fracchina, T. W. Calvert, and M. S. Draw. Deriving spectra from colors and rendering interference. *IEEE Computer Graphics and Applications*, 19(4):61–67, 1999.

[8] M. Sunkel, J. Kautz, and H.-P. Seidel. Rendering and simulation of liquid foams. In *Vision, Modelling and Visualization*, 2004.

[9] L. Szirmay-Kalos, B. Aszodi, I. Lazanyi, and M. Premecz. Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum*, 24(3):695–704, 2005.

A Constraint Based System to Populate Procedurally Modeled Cities with Buildings

Johannes Scharl*
Supervised by: Daniel Scherzer†

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Austria

Abstract

Creating large-scale virtual environments for interactive applications such as computer games poses a demanding challenge for computer graphics. We present a system that procedurally creates urban environments including street networks, street geometry and building parcels. Our main contribution is a constraint based system that chooses the "best fitting" building for every parcel from a set of existing buildings. Building properties such as the footprint, area, and faces that should have street access are used to select the most suitable building.

Furthermore we introduce a robust technique to create 3D street geometry for streets that adapt to different terrain heights and describe a method to create a more realistic city shape and more detailed outer regions.

Keywords: Procedural Modeling, Urban Environments, Computer Games

1 Introduction

The usual way to create urban environments for computer games or movies is to use commercial modeling packages like Autodesk Maya. This is a very time consuming, tedious and expensive task and gets less and less suitable for modern applications that demand even larger and more detailed environments. A promising approach that has emerged in recent years is to create content procedurally.

Urban environments are mainly defined by their street network. Such a network forms the back bone of a city and determines its layout. Therefore it is the first thing that has to be generated when modeling a city.

Recent state-of-the-art techniques [10, 15] rely on extended L-systems to create such networks. Usually a top-down approach is used, meaning that major roads are created first, because they define the main routes and districts in the city. Regions surrounded with major roads are then filled by minor roads, creating the finer structures of districts and neighborhoods. When using this approach, areas surrounded by major roads are usually located at the city center. This often leads to sparse regions at the outskirts of the city, where no minor roads can be created. We propose an approach to generate cities where minor roads are also generated in the outer regions. This is illustrated in figure 1.

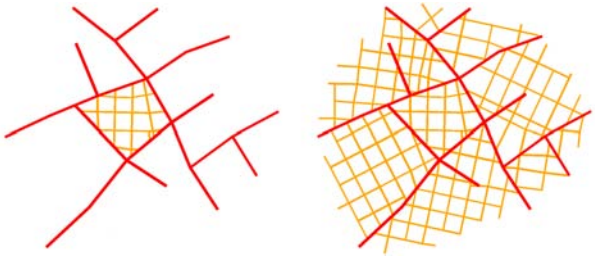


Figure 1: Left: In previous methods, no detailed neighborhoods are created in the outskirts of the city. Right: Outer regions created with our method.

After the street network is generated, street geometry is created and the blocks in between are subdivided into building parcels. Tessellating the street geometry is straightforward in a case where all streets are in a single plane, but gets complicated if streets adapt to three dimen-

*johannes@cg.tuwien.ac.at
†scherzer@cg.tuwien.ac.at

sional terrain. In this case, junctions have to be kept planar. This is usually handled by forcing the junction geometry to be parallel to the ground plane, resulting in unnatural steps in steeper roads. We introduce a way to adjust these junctions to adapt them to the underlying terrain.

Buildings are an essential part of every urban environment. Usually each model is hand-crafted in a modeling software and placed at its destination by a designer or artist. This approach is not feasible for larger urban environments. Typically only certain regions of the city are crucial for a computer game, while other regions do not contain a lot of individual detail. A method that fills these regions automatically greatly reduces the time needed to add building models to a road network. As our main contribution, we propose a method to select a building from a set of existing models that fits best for a certain building parcel and place it there.

2 Related Work

Our work is based on previous procedural modeling methods that employ *L-systems*. L-systems where originally developed as a formalism for plant modeling [12].

An extension to L-systems that allowed plants to communicate bidirectionally with their environment, called *Open L-systems*, was introduced later by Měch et. al. [9]. This method was developed further by Parish et. al. [10] to create city street maps using a set of production rules.

Weber et. al. [15] extended this method to simulate a three-dimensional urban model over time. They define a city hierarchy that guides the creation of such networks. Because we use this hierarchy definition, we will discuss it in greater detail in Section 3.1.1.

The CityEngine [11] is a commercial software package that is capable of procedurally generating complete cities. Input is provided in the form of many controllable parameters and various image maps: height maps can be used to model terrain, obstacle maps denote regions where no streets should be created and population density maps control the type and density of the streets and buildings in certain regions. The system is capable of creating large street networks, building parcels and even buildings. Streets are generated using L-systems, while buildings are created with a shape grammar technique.

Other methods to create street layouts and networks include interactive editing of a tensor field [2], a mixture of interactive and procedural techniques where main streets have to be created manually, while minor roads are generated automatically [5], and image based approaches that rely on aerial images to reconstruct a road network [1]

Recently, procedural generation of buildings and facades has been researched heavily, including the generation of facades using shape grammars [8, 16] and interactive editing of shape grammar rules [7].

An excellent review of various urban modeling techniques can be found in a recent survey paper by Vanegas et. al. [14].

Gebhart [3] describes a system that helps artists to create 3D street networks. Streets are "drawn" by an artist or designer, and detailed geometry is created semi-automatically by setting parameters such as the number of lanes, the radius of a curve, etc. Zimmermann [17] presented a technique to construct a fully polygonal 3D street representation out of centerlines. Although both methods produce visually impressive results, both fail to address the problem of maintaining a stable and realistic tessellation for 3D streets that adjust to terrain levels of different heights.

The problem of finding a model that fits into a certain environment has not been investigated very extensively. Kjølaas [6] presented a system that automatically places furniture into a given floor plan by selecting a template from a given set of default templates for common rooms and adapting it to the given room.

In the field of automated building placement a lot of work has been done in the direction of recognition and reconstruction of buildings from aerial images [4, 13]. This is usually done in the process of reconstructing existing cities, e.g. for mapping applications, but not for artist-created urban environments.

3 Our System

In this section we present our techniques to create street networks for urban environments, as well as a method to tessellate the street geometry in a simple and stable way. Additionally, we propose a technique to match the "most suitable" building to a parcel from a set of previously modeled buildings.

3.1 The Street Network

Our algorithm to create street networks is based on the work of Parish et. al [10] and Weber et. al. [15]. Streets are created using a system similar to extended L-Systems, although we chose not to implement a string rewriting system, but to apply the production rules directly to the street objects to avoid slow string operations, as proposed in [15]. We will first explain our city hierarchy definition in Section 3.1.1 and describe a set of important control parameters that are used for creating streets in Section 3.1.2. Finally, we will describe the algorithm based on the described hierarchy in Section 3.1.3.

3.1.1 City Hierarchy

We use a city hierarchy definition similar to the one introduced by Weber et. al. [15]:

A *street network* is a planar graph (V,E) with nodes V and edges E. A street consists of one or more edges $e \in E$, the *street segments*. Each street segment e connects two nodes $n(e)_1, n(e)_2 \in V^2$. Streets can be *major* or *minor*

streets and have different *widths*. A facet in the planar graph (V_{major}, E_{major}) is referred to as *quarter*, that is an area surrounded by major roads. A facet in (V, E) (an area surrounded by any street) is called a *block*. Each block can be subdivided into *building parcels*. This hierarchy is illustrated in Figure 2.

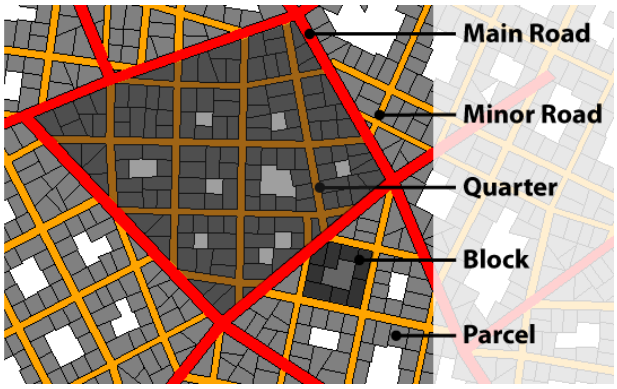


Figure 2: City Hierarchy. *Main roads* are displayed red, *minor roads* are orange. *Quarters* are areas surrounded by major roads. *Blocks* are surrounded by any road and divided into *parcels*.

To create a city obeying this hierarchy, major streets are created first, and spaces in between are filled with minor roads afterwards. This results in a planar street network and buildings blocks in between. A block consists, according to our hierarchy definition, of multiple building parcels, each defining the space a single building can take up. These parcels can be generated by repeatedly subdividing a building block.

3.1.2 Control Parameters

The creation of streets can be controlled with a lot of different parameters:

- Cities usually have different layouts. Streets in New York City are strictly rectangular, whereas Paris follows a loosely circular pattern. Cities with no superimposed pattern grow organically. In our system, major and minor roads can follow *different patterns*.
- Street *length*, *width* and *angles between adjacent street segments* can be controlled.
- To avoid street ends near existing junctions, a distance `snappingDistance` can be defined. If the distance between a street end and a junction is smaller than `snappingDistance`, the street end is snapped to this junction.
- A *height map* may be specified to create a terrain (see Figure 3). The streets will then adjust to this terrain. If the slope of the street is larger than a user defined threshold `maxSlope`, the street is rotated until it is plain enough, or removed if that is not possible.
- Urban environments may contain areas where no streets should be created, such as parks or water. Such areas can be denoted in an *obstacle map* (see Figure 3). This map is sampled regularly and streets will avoid any obstacles.
- Real cities grow after demand: Major roads connect centers of high population densities, while minor roads provide access to the major roads in populated areas. A *population density map* (Figure 3) can be set to control the development of major and minor streets.
- The average size of a building parcel can be controlled to adjust the parcels to the desired building size.

All of these parameters can be set using the user interface of our application.

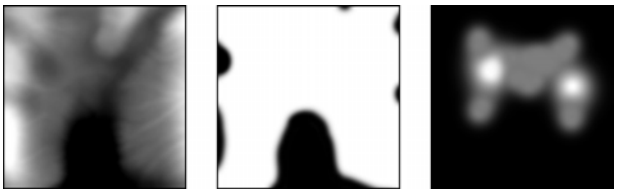


Figure 3: Input maps for a bay area environment. From left to right: (1) Height map, (2) Obstacle map, (3) Population density map.

3.1.3 Creating the Street Network

As explained above, the algorithm is divided in two stages: (1) creating major streets and (2) identifying quarters surrounded by major streets and filling them with minor streets. Quarters and blocks can be identified easily using a planar graph face traversal algorithm.

As discussed in [10] we use a 3-level hierarchy to evaluate parameters for a new street segment: First, an *ideal successor* is created. This is a new street segment without any parameters assigned. For this new street segment, the *global goals function* is evaluated. The location and orientation of the street is set according to the superimposed street pattern and the local population density. After the initial parameters have been evaluated, the street segment is adapted to its local environment by calling the *local constraints function*: This function changes the location and orientation of the street according to the parameters in Section 3.1.2: The new street segment is snapped to existing junctions, the street is adjusted to the local terrain slope or changed to avoid any obstacles such as parks. The procedure is the same for major and minor streets, although different parameter sets can be used.

One of the main problems of this approach is that minor streets are only created inside quarters that are surrounded by major streets. This leads to unrealistic results at the

boundaries of the street network, where no minor streets are created.

We have solved this problem by calculating the convex hull around the whole street network and using this hull to create additional quarters at the city boundaries.

If the points of the convex hull are connected by a straight line, outer regions of the city look ”cut off” and unnatural. To account for that, we bulge the hull by random amounts to create a more realistic city shape.

After all streets have been created, we use the same algorithm we used before for identifying quarters to now find all blocks surrounded by streets (major and minor). To create parcels of the size the user specified, each block is recursively subdivided into building parcels using a method similar as the one described in [15]: (1) Select the longest side of the polygon and create a perpendicular split line at its center. (2) Split the polygon along this line. (3) Repeat this for each new polygon if its area is larger than a predefined threshold `maxArea`.

In most cases, only parcels with street access should be created, because most buildings have an entrance that should face a street. If the boolean parameter `deleteInnerLots` is set, we delete all parcels that have no direct street access. The algorithm is illustrated in Figure 4.

The complete street creation process is illustrated in figure 5.

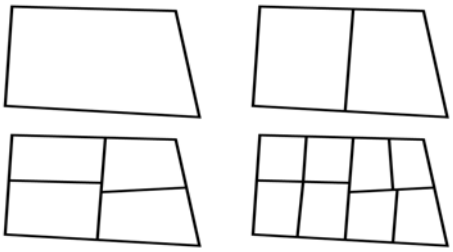


Figure 4: This figure illustrates how a block is recursively split into parcels until each parcel is below the area threshold `maxArea`.

3.2 Street Geometry Tessellation

Street tessellation is not the main scope of our work, so we wanted to implement a simple, but stable geometrical representation of the street network that enables simple shading of streets that adjust to the slope of the underlying terrain.

Tessellating a single street is straightforward, but some problems arise at junctions: If each street would be tessellated and rendered independently, discontinuities and z-fighting would appear where two streets meet. Therefore we need a special geometric representation for junctions.

Another problem that arises is how streets that adjust to multiple height levels should be tessellated so that junctions are planar, but do not form unnatural steps on slopes.

We will first describe our method for street tessellation in case of streets that are coplanar in Section 3.2.1. In Section 3.2.2, we will discuss how we solved the problem of non-coplanar street networks.

3.2.1 Geometry for Planar Streets

The street network is represented as a planar graph of edges that connect to each other at junctions. To draw this network, we need to construct a fully polygonal street representation. The original edges serve as centerlines for the street geometry.

The geometric representation of a street network consists of *junctions* and *street segments*. Each street has a certain `streetWidth` that was set in the street network creation process. We call the point where two street centerlines meet the *center point* of a junction.

To create a polygonal representation of the street, we use a similar approach as the one discussed in [17]: we offset lines from the centerline on both sides by $\frac{\texttt{streetWidth}}{2}$. These offset lines are called *street outlines*. Street outlines of two adjacent street segments intersect at the *corner points* that define the corners of the junction and separate the junction geometry from the street segment geometry. To create the junction geometry, the two corner points of one end of a street segment form a triangle together with the junction center point. This triangle is called a *street head*.

A junction consists of n street heads, where n is the number of street segments adjacent to the junction. This is illustrated in Figure 6. In cases where two junctions are too near to each other, so that junction geometries overlap, the two center points are merged to create a stable junction geometry.

Our method produces simple, but stable results, as can be seen in Figure 7.

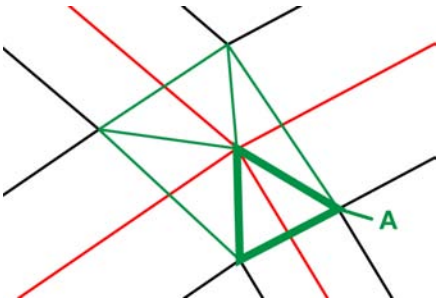


Figure 6: This figure illustrates how a junction is defined by 4 street heads. The street centerlines are displayed in red and meet at the center point. Street outlines are colored black. Street heads are shown in green, the lower right street head is highlighted for clarity (A).

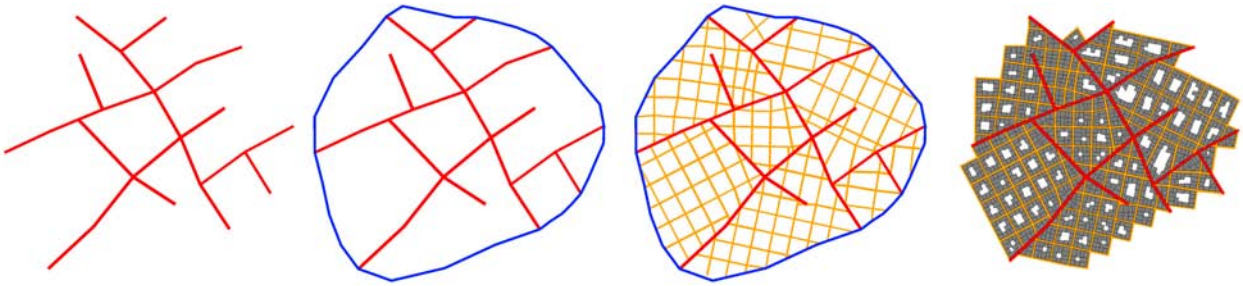


Figure 5: This figure illustrates the steps necessary to create a street network. From left to right: (1) Major streets are created (organic pattern, red). (2) the convex hull is calculated and bulged (blue). (3) Minor streets are created inside the quarters (grid pattern, orange) (4) The final street network with building parcels, convex hull and dead end roads removed.

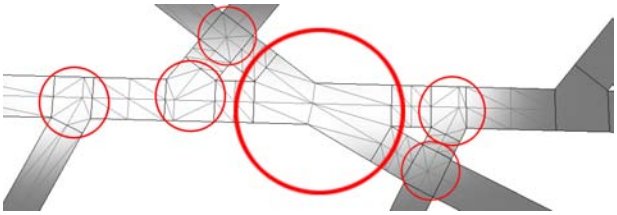


Figure 7: Even complex junctions are tessellated correctly with our method.

3.2.2 Geometry Displacement for 3D Streets

The method described in [17] creates good results as long as all streets are in one plane. When streets are not coplanar, a number of problems arise: The junction geometries described in Section 3.2.1 need to be flat, otherwise streets will twist unnaturally. We introduce a method that nestles street segments and junctions to the terrain underneath while preserving the planarity of junction geometries.

To create flat junction geometries, initially all the vertices of the junction geometry are placed at the same height as the center point of the junction. This leads to unaesthetic and unrealistic steps and extreme slopes, as can be seen in Figure 8. These steps can be smoothed by moving the junction geometry into the tangent plane to the terrain surface at the junction center point. This can be done by calculating how much the normal of the terrain is rotated against the up vector. Based on that, a rotation matrix is calculated around an arbitrary axis that is later applied to every vertex of the junction. As a result, the whole junction geometry is rotated into the tangent plane of the terrain surface. An illustration can be found in Figure 8.

Unnatural steps in steep streets are avoided this way, but we introduce a certain lateral grade. This lateral grade is limited by the longitudinal slope of all other streets adjacent to this junction. This is acceptable for interactive applications such as games, since the maximum allowed longitudinal slope (12%) is not much higher than the maximum allowed lateral grade (8%)[1].

We follow the approach in [17], where street segment

[1] In Austria, this may differ in other countries, and for mountain roads.

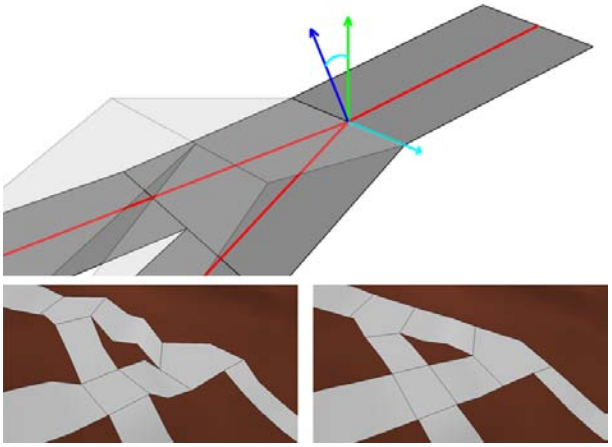


Figure 8: Smoothing of junction geometry. Top: The initial geometry is displayed transparent. It is then rotated into the tangent plane of the terrain surface. Bottom, left: Geometry before rotation. Right: After rotation.

ends are modified so that they connect to the junction at a line perpendicular to the direction of the segment. This is done for the following reason: If the two corner points of a street head do not lie on a line perpendicular to the street segment direction, the street may twist unnaturally or become uneven. Refer to Figure 9 for an illustration.

In between the street heads, the geometry of the street segment is subdivided regularly and displaced according to the terrain height to nestle against the underlying terrain surface.

3.3 Building Assignments

Our main contribution is a technique to assign buildings to parcels from a set of previously modeled buildings. After the street network creation process described in Section 3.1, building parcels of various size and shape have been created, most of them being rectangular. Modeling a suitable building for all of them by hand would be a huge and tedious effort. We propose a method that selects the ”best fitting” model for each parcel from a set of existing buildings. By ”best fitting”, we mean the model that

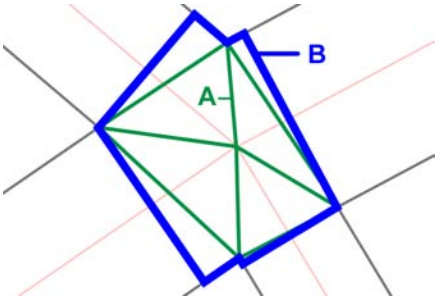


Figure 9: Street ends are modified to create connecting lines perpendicular to the street segment direction. The original junction geometry is shown in green (A), the modified geometry in blue (B).

occupies most of the building parcel, while satisfying various constraints, such as not protruding from the parcel. These buildings can be created using a commercial modeling software like Autodesk Maya, or may be procedurally generated.

The rest of this section is structured as follows: we first discuss some characteristics of building models in Section 3.3.1. Then we describe our method for selecting the "best fitting" building for a parcel in Section 3.3.2.

3.3.1 Building Properties

Each house has a footprint that can be defined as the convex hull of all vertices in its geometry projected onto the ground plane. This gives a good estimate for the area the building will need on a parcel. If this area exceeds the area of a certain parcel, the building will not be considered further for it (Constraint 1).

A building consists of sides that have to face a street side, e.g. because there is a door that needs street access. Also, there are sides that must not face a street, e.g. because there is a plain brick wall on this side. We refer to them as *Street Access Sides* and *Inaccessible Sides*. These sides pose constraints for our system that have to be met. *Street Access Sides* have to be aligned and placed next to streets to guarantee direct street access (Constraint 2), and *Inaccessible Sides* have to point away from streets so that they are not directly visible (Constraint 3). All other faces of the building may face a street, but they do not have to. See Figure 10.

All these properties are stored as meta information in a XML file for each building model.

3.3.2 Selecting a Building

The building with the largest footprint that meets all the criteria described above will be selected as the "best fitting" building for a parcel.

The set of previously modeled buildings is stored in a list ordered by footprint area size from largest to smallest. This list is enumerated for each parcel. All models

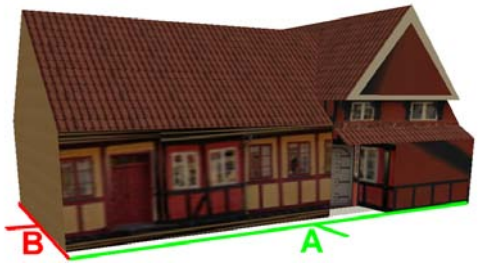


Figure 10: A simple building model. *Street Access Sides* are displayed by green lines on the ground plane (A), red lines denote *Inaccessible Sides* (B).

that have a larger footprint area than the current parcel are discarded. Also, models that contain more *Street Access Sides* than the parcel has adjacent street sides are not considered, because it is not possible to align them all correctly. For every remaining building, a series of transformations and tests are applied. The first model that passes all the tests is chosen for the current parcel. This guarantees that the building that occupies most of the parcel area while meeting all constraints is selected. The process is illustrated in Figure 11.

- The building footprint is moved into the center of the parcel.
- The largest *Street Access Side* of the footprint is aligned to the largest side of the parcel that is adjacent to a street to get an initial alignment for the building. This suffices for most of the buildings, since many of them only have one front side that needs to face the street.
- Rotate the building footprint until all *Street Access Sides* face a street and all *Inaccessible Sides* do not. A side faces a street if it is nearly parallel[2] and a ray cast perpendicular from its center directly hits a street.
- Move the footprint as near as possible to any streets adjacent to the parcel. Buildings usually adjoin directly to streets, but a minimum distance can be configured in the user interface.
- Check if all points of the footprint are inside the parcel.
- If any of the above tests failed, repeat the process with the next smaller building. If a valid solution was found, assign the building to this lot considering the found transformations.

If the parcel is located on a slope, the building is moved down so that every vertex is on the ground or beneath. If the slope of the gradient of the parcel exceeds a user defined threshold `maxParcelSlope`, the parcel is discarded and no building is assigned.

[2] we chose a max. deviation of $\pm$ 30 degrees

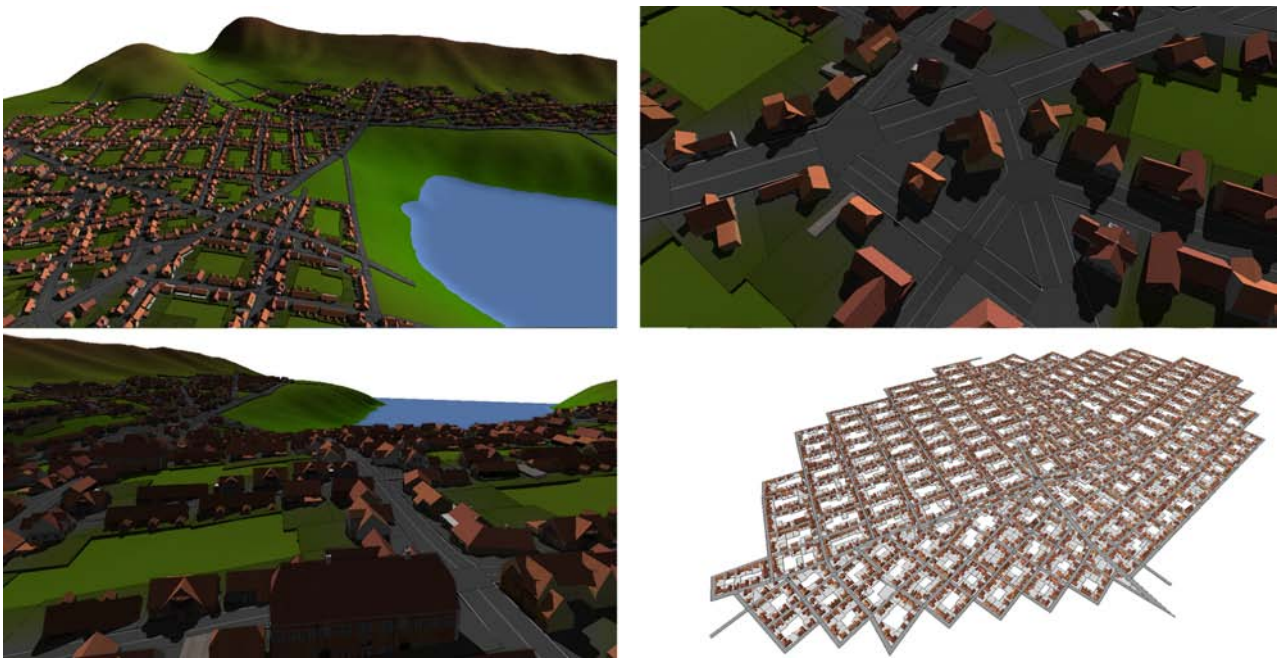


Figure 12: Some results from our system. Top left: a small village on a hillside next to a bay. Note how the roads adapt to different terrain heights and stop if they become too steep. If the ground is too uneven, no parcels are created. Top right: Close-up of a more complex junction in the village. Bottom left: Close-up with view over the bay to the hillside where streets spread over the plain of the terrace. Bottom right: a larger city without terrain with far over 100 streets and more than 2.500 buildings.

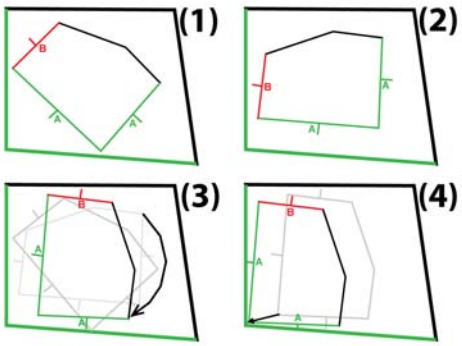


Figure 11: Steps for fitting a building footprint into a parcel. The outer polygon illustrates the parcel, the inner one the building. *Street Access Sides* and parcel sides adjacent to a street are colored green (A), *Inaccessible Sides* red (B). (1) The footprint is moved into the center of the parcel. (2) The largest *Street Access Side* is aligned with the largest parcel side with street access. (3) The footprint is rotated. (4) The footprint is moved near to the streets.

4 Results

Our system is capable of procedurally generating whole cities with dozens of streets and hundreds of buildings within seconds. Most parameters used for the creation of street networks and the assignment of buildings can be changed in our user interface. The assignment of buildings worked well in our tests, however the visual result depends on the quality and amount of models used. If only a few buildings are available, very uniform neighborhoods are created. The larger the set of models is, the more diverse the cities become. For our tests, we used 20 models available freely at Google 3D Warehouse and achieved pleasing results. Some screen shots of our results can be seen in Figure 12.

We implemented our system in C# using the XNA framework for rendering. All tests and images in this paper were made on a system consisting of a Core 2 Quad 6600 with 4GB RAM. At the moment our system is single threaded, but it could easily use multi threading, especially for the parcel subdivision and building assignments. See Table 1 for detailed results.

Task	Count	Time
Street creation	73 streets, 283 segments	0.65s
Street tessellation	73 streets, 283 segments	0.17s
Parcel Creation	811 parcels, 107 blocks	0.08s
Buildings	776 buildings assigned	2.24s
Total		3.14s

Table 1: Performance benchmarks of our system for a small city consisting of 4 main and 69 minor roads. The whole city was created in just over 3 seconds.

5 Conclusion and Future Work

We presented a system that procedurally creates urban environments including street networks, street geometry and building parcels. Very different types of cities can be created by changing parameters in the graphical user interface. Our main contribution is a constraint based system that choses the "best fitting" building for every parcel. Building properties such as the footprint, area and faces that should have street access are used to select the most suitable building.

Furthermore we describe for the first time a robust method to create 3D street geometry for streets that adapt to different terrain heights.

In existing street generation systems, no minor roads are created at the city borders. We addressed that problem by calculating a convex hull around the city and widen it by random amounts to create a more realistic city shape and more detailed outskirts.

5.1 Future Work

Image maps could be used to control more parameters of the building assignment process and give the user more control over the selection of buildings for the parcels.

An input map similar to a height map could be used to control the building height for different regions. Another map could be used to manage the type of buildings: A building can be of a certain type (residential, industrial, suburban, etc.), and each type is associated with a certain color in the map. This information could be used in the building assignment process to create districts with a different look and feel.

The main limitation of our current system is that it is static. Models placed automatically can not be moved or modified in any way. We want to give the user the power and flexibility to change the environment after it has been created automatically.

References

[1] D. G. Aliaga, C. A. Vanegas, and B. Beneš. Interactive example-based urban layout synthesis. *ACM Trans. Graph.*, 27(5):1–10, 2008.

[2] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang. Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3):1–10, 2008.

[3] Gernot G. Automatisches generieren von 3d-straßensystemen. Master's thesis, Vienna University of Technology, March 2008.

[4] C. Jaynes, E. Riseman, and A. Hanson. Recognition and reconstruction of buildings from multiple aerial images. *Comput. Vis. Image Underst.*, 90(1):68–98, 2003.

[5] G. Kelly and H. McCabe. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology*, pages 8–16, 2007.

[6] K. A. H. Kjølaas. Automatic furniture population of large architectural models. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, January 2000.

[7] M. Lipp, P. Wonka, and M. Wimmer. Interactive visual editing of grammars for procedural architecture. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, New York, NY, USA, 2008. ACM.

[8] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[9] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 397–410, NY, USA, 1996. ACM.

[10] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 301–308, August 2001.

[11] Procedural Inc. Cityengine, www.procedural.com, 2010.

[12] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[13] I. Suveg and G. Vosselman. Reconstruction of 3d building models from aerial images and maps. *ISPRS Journal of Photogrammetry and Remote Sensing*, 58(3-4):202 – 224, 2004.

[14] C. Vanegas, D. Aliaga, P. Wonka, P. Müller, P. Waddell, and B. Watson. Modeling the appearance and behavior of urban spaces. *Comput. Graph. Forum*. to appear.

[15] B. Weber, P. Müller, P. Wonka, and . Gross. Interactive geometric simulation of 4d cities. *Computer Graphics Forum*, 28(2):481–492, April 2009.

[16] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003.

[17] M. Zimmermann. Procedural construction of streets. Technical report, ETH Zürich, 2007.

Data Acquisition

Laser Scanning Versus Photogrammetry Combined with Manual Post-modeling in Stecak Digitization

Goran Radosevic
Supervised by: dr Selma Rizvic
Faculty of Electrical Engineering Sarajevo
Bosnia and Herzegovina

Abstract

Stecci (sing. stecak) are hand carved medieval Bosnian gravestones considered as valuable cultural heritage objects. One of the best ways for their preservation is digitization.

In this paper we compare digitization results and procedures for one of the most famous stecci – The Stecak from Donja Zgosca. The object was first digitized using a Minolta 910 laser scanner. Later we created the 3D model from photos using photogrammetry and improved it in 3ds max. We present advantages and drawbacks of these two procedures and characteristics of the obtained models. Results of this comparison will be used in future digitization projects.

Keywords: Laser Scanning, Photogrammetry, Manual Modeling, Stecak, Cultural Heritage, 3D Modeling.

1 Introduction

Cultural heritage within its set of materiality, traditions and knowledge helps us to better understand the past itself. Therefore it is very important to preserve these monuments in a way we see them now for next generations. Thus, new technologies can be very helpful. Today we are able to create virtual model of a real object using various techniques. We used two different techniques in our research, expensive technique of laser scanning and much cheaper, but also good technique for acquisition of 3D models from 2D images, photogrammetry. We also made a step forward and introduced some improvements of the model achieved using photogrammetry, as we will see in more details later in this paper.

Figure 1: The Stecak from Donja Zgosca

Stecci are hand carved medieval Bosnian gravestones. We applied both techniques on the Stecak from Donja Zgosca, and got some interesting results. This stecak originates from the second half of 14th century. It has a great importance for Bosnian history because it is assumed that the Bosnian king Stjepan II, who died in 1353, was buried under this stecak [1]. This monument is currently located in the botanical garden of the BH National Museum in Sarajevo (Figure 1).

In many cases, like with the Stecak from Donja Zgosca, the traditional modeling (for example manual modeling using 3ds Max or Maya) would require much more work and effort, and the final result would not be satisfactory. The model created using these methods would not be sufficiently accurate. It would not contain enough information about the real object. This is why we use methods such as 3D laser scanning, which produces a virtual model with high accuracy. In the first part of our research we used a Minolta 910 laser scanner (Figure 2), borrowed from our project partners, the University of Bristol, UK.

Figure 2: Scanning the stecak with a Minolta 910 laser scanner

In the second part of our research, we used photogrammetry technique for virtual reconstruction of the stecak. The basic principles of photogrametry are briefly presented in the Section 4. In this project we used the Photomodeler software for implementation of photogrammetry technique with manual approach of creating 3D model. The model was later improved in 3ds max software.

The paper is organized as follows: the Section 2 gives a short overview of the related work in similar projects, in the Section 3 we describe the laser scanning approach, applied on the stecak. In the Section 4 we describe the process pipeline of photogrammetry combined with manual 3D post-modeling of the model, and steps for its implementation. In the Section 5 we compare results achieved using these two approaches, and describe

advantages and drawbacks of both techniques. In the Section 6 we present conclusions based on our experience from this project, and illustrate reasons why to use one technique instead of another in future projects.

2 Related work

When creating 3D model of a real object, it is very important to choose the most appropriate technology and procedures that can create the best final output in accordance with the project's specifications and requirements.

Laser scanning shows its full potential in open pit mining environment application, where no rival technology comes close to matching the utility of a laser scanner, not even digital photogrammetry [11]. In project of recording 3D measurements from medieval castle of Haut-Andlau (Alsace, France) [12] laser scanning and photogrammetry were used. Application of each one of them resulted with similar level of satisfactory accuracy. Main difference between these two approaches is that laser scanning is focused on grid of points, without taking specific object structures into account, like corners or edges. On the other hand, photogrammetric measurements concentrate on object discontinuities and representative structures, even without generating dense point cloud [12]. If our method of improving the generated 3D model in 3ds Max had been used in this project, the achieved realism of the castle could have been even better and mainly flat surfaces could have had more details.

The accuracy of created 3D points in both techniques was compared in application on the ancient church of Pozzoveggiani, Italy [13]. Here the photogrammetry technique has given the similar or even better results than laser scanning technique, but for the best results author adviced using some combination of these techniques, as each one has attributes and elements that complement one another. In addition our method could be used for increasing geometry details on the model of the church created using photogrammetry technique.

Today's laser scanning technique offers a good way for 3D model acquisition, but also has a lot of issues which are preventing its wider use [4]. Some projects are introduced, which could boost the diffusion and evolution of 3D scanning technology [4]. New tools and solutions for improving these techniques are often introduced, such as TexAlign, which helps the user to improve image-to-image correspondences and it is presented and applied on the model of David's statue. Another example is a new solution for generating 3D models from high resolution photos, which is presented on the model of Arc du Triomphe, Paris, France [4]. This solution uses a special algorithm to calculate accurate surface details achieved by triangulation. Our method of improving the model by estimating surface details using object's textures is not that accurate.

3 Laser scanning

Laser scanners provide a method of capturing accurate information about object's surfaces. The stecak was scanned with a Minolta 910 laser scanner. It is a scanner for close range and indoor applications [2]. This scanner has accuracy of less than a millimeter.

The scanning is based on the principle of laser triangulation, Figure 3. The target is scanned with laser beams. The laser scanning mechanism characterizes each point on the scanned object according to its location in 3D-space by scanning the surface of an object with one focused beam, and recording the reflected light using CCD camera. Each point on the object is described by 3 numeric values which correspond to 3D coordinates X, Y, and Z.

The X coordinate of each point on the object is calculated from an accurate measurement of the position of the scanning mirror in the camera. The Y coordinate is calculated from an accurate measurement of the camera motion system (CMM). The Z, or range coordinate, is calculated through laser triangulation within the camera. Surface shape measurements of the object are obtained through triangulation, and then converted into a 3D polygonal mesh [3].

The scanner measures 640 x 480 points regions within one scan, simultaneously acquiring surface shape data and color image data.

After measuring the 3D depth data, the Vivid 910 uses its CCD to capture a 2D image in the same way as a digital camera. The CCD relies on ambient light to illuminate the target. The scanner software then matches points on the photograph to points in the surface mesh and exports the data as a CDM file which contains both the mesh and bitmap. In addition, a color image of the object can be also obtained by scanning the CCD through a RGB filter while the stripe light is not emitted. (A band pass filter is used when the stripe light is emitted.)

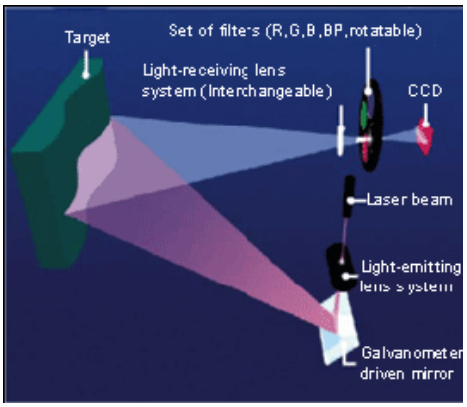


Figure 3: Minolta Vivid 910 measurement principle

Since the less illuminated scan areas produce better results, the model of stecak was scanned during the night because of the intense light in that part of the Museum's botanical garden during daylight. Given that, the textures produced were not satisfactory (Figure 4, left), so the

decision was made to continue without the original textures. Instead, the model was assigned with the appropriate textures later, during its reconstruction in Maya.

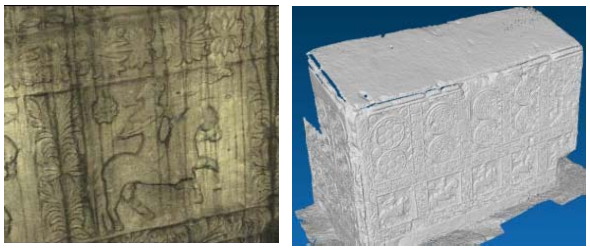


Figure 4: Texture of the model captured by the laser scanner (left); the stecak without textures. Multiple scans merged into a single using the Stitcher (right)

Individual scans, created by laser scanning method, were later connected together in a polygonal mesh by using the Stitcher software tool provided with the laser scanner (Figure 4, right). This software was used for editing captured scan data, merging scans into single "watertight" mesh, and then exporting .OBJ file to Maya.

The computer model of the scanned stecak was transferred from Stitcher to Maya. The size of the laser scanned model was so large that it needed to be reduced to 5% quality in order to manipulate it in Maya.

Maya has a function for reducing number of vertices of the model, optimizing it for easier use. Problem is that this function works only for models created in Maya, but not for polygon meshes generated by the scanner. The only suitable solution was to load the original scan file to Stitcher, and do all reducing there. After applying new modifications in Stitcher, reduced model was exported from there, and imported to Maya. Geometry of the stecak was here improved in some areas, and the damaged part in the lower corner was repaired (Figure 5).

Radiance – the physically based rendering software system was used for calculating light over the object. In Maya the new material was created with information about the object's radiance. That material was applied on the object.

The finished model's geometry was exported in an OBJ file, and TIFF texture file was created by the material conversion as well.

After that, we further optimized this model in Meshlab tool [4, 5]. Results of comparison of various optimization methods are presented in Section 4.

Figure 5: Comparison between raw model of the stecak before reconstruction (left); Final model of the stecak after repairing in Maya (right)

The obtained model was used for creating the sun simulation animation [2] and in the online application "Virtual Sarajevo" [7]. We also used this model without optimization to introduce the archaeologists from the National Museum of Bosnia and Herzegovina with possibilities in virtual reconstruction of the cultural heritage objects, reconstructing the damaged part of the object in Maya (Figure 5).

4 Improved photogrammetry

Our goal in this phase of our research was to create a model of stecak which has the level of realism similar to the model created using laser scanning technique, using much cheaper equipment. The process of creating 3D model using improved photogrammetry consisted from three steps:

- Taking photos
- Creating model in Photomodeler
- Improving model in 3ds max

4.1 Taking photos

The first task that had to be done within the photogrammetry process pipeline was taking photos. It is very important to have knowledge about the photography technique and the camera parameters which are used in this process. To obtain high accuracy and reliability, the photos must be of the highest quality. The photos taken on the site are shown in Figure 6. We used Canon PowerShot Pro 1 camera for taking the photos.

Figure 6(a): The photos used to recover 3D

Figure 6(b): The photos used for capturing textures from the sides of the object

Figure 6(c): Additional photos used for obtaining more information from the top sides of the object

The photos must be taken in a precisely defined manner in order to be used later in a photogrammetry process.

Figure 6(d): Additional photos used for recovering textures and geometry from the damaged part of the object

This process requires that certain parameters of the photos remain unchanged. Focal length must be the same in all taken photos. Digital camera must have the option for manual adjustment of the focus or for locking the focal length. After adjusting the focus in a way that the image is sharp, the same focus value should be locked for every picture taken in the set.

Other parameters that must remain unchanged while taking photos are: image resolution, zoom, camera distance from the object, exposition. Brightness and possible shadows should be the same in all taken pictures in the set. If not, we can have problems in overlapping textures that look different when taken from different positions [4]. Flashbulb can be used, but we found that pictures look more natural when they are taken in daylight.

It is a good practice to take more photos from the same camera position, because it is not hard to do, and it can save our time if we find that some of the taken photos are not good enough. The tripod should be used if it is available.

Camera calibration is a procedure of taking photos of Photomodeler's calibration grid (Figure 7) with the same camera that was used for taking photos on the site. The final goal of camera calibration process is to introduce Photomodeler with internal camera parameters: focal length, format size of the sensor, image size or lens distortion. After taking calibration photos, they should be imported in Photomodeler and then automatically processed by this application. After successfully finished calibration process, Photomodeler will use internal camera parameters for later calculations of the taken pictures based on triangulation.

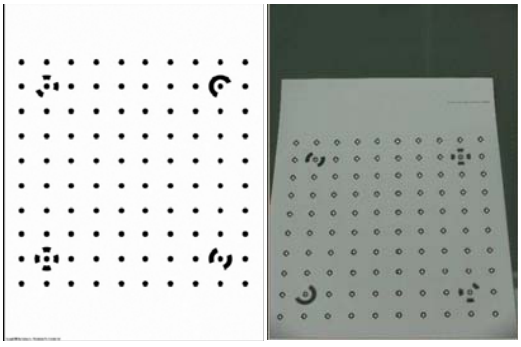


Figure 7: Photomodeler's calibration grid: Original paper (left); Photo with generated points after processing (right)

In this project we did not use camera calibration process because we wanted to see if it is possible to create satisfying model quality without it. We achieved almost identical results in this way as in a project where calibrated camera is used, because Total Error is very low indicating that estimated camera parameters are excellent.

4.2 Creating the model in Photomodeler

Model creation in Photomodeler is an iterative process. Each iteration consists of the following steps:

- Marking important elements on the photos
- Referencing elements between different photos
- Starting Photomodeler's automatic processing
- Drawing surfaces in 3D viewer

In the marking process we should mark the well visible elements (points, edges, curves, lines, etc.) that we can see at two or more photos. Every element must be marked in minimum two pictures in order to be processed and to reveal its 3D information. It is very important to have good quality photos with high resolution. Lower quality photos could lead to lower accuracy of the final 3D model.

Referencing step refers to connecting the same elements on different photos. In this way, Photomodeler "knows" that the same element is appearing in different photos. This information is used in automatic processing step in which 3D information can be calculated using triangulation.

After that, we can start Photomodeler's automatic process of calculating 3D data. If the processing step is finished successfully, the 3D model from given elements is created. We can make iterations of the process improving its elements until the Total Error is minimal (Figure 8). After the processing step, the 3D model is generated, and we can see it in the 3D viewer. In the 3D viewer we can also draw all surfaces.

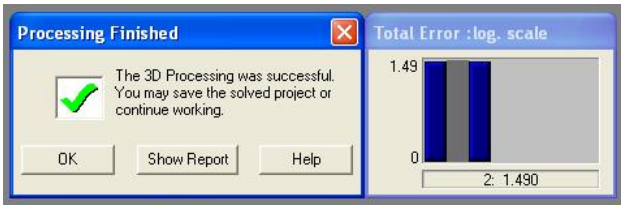


Figure 8: Notification for successfully finished processing. Processing was done in two iterations of improving data, with total error of 1.49

Now we can start with the next iteration of modeling by repeating these steps of marking and referencing elements, processing and drawing surfaces in the 3D viewer. We should repeat this cycle until we have generated the 3D model with desirable level of accuracy.

Without using calibration in the beginning, accuracy of our generated 3D model is very low. The software has no information about the camera parameters, thus 3D coordinates could be calculated in a wrong way (Figure 9).

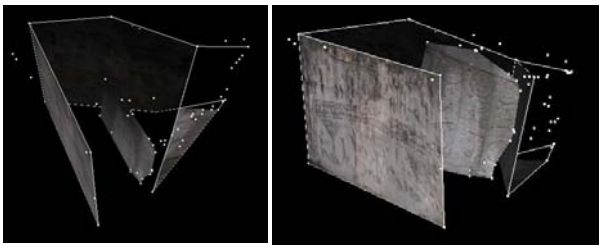


Figure 9: Due to the wrong calibration parameters and lack of marked information in the beginning, Photomodeler generated the wrong 3D information

In our example we have created a project with estimated camera parameters, which could be wrong, and we got poor results after processing. Lack of camera parameters information could be compensated by high quality photos, and well marked and referenced elements. In the beginning this can be very hard work that does not give us the results we expect.

The Point Table is used for approving overall accuracy of generated 3D information (Figure 10).

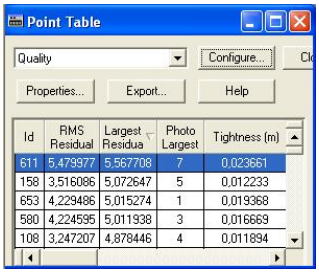


Figure 10: The Point Table in Photomodeler is used to find points that are not marked correctly

The Point Table holds all information of all points marked in our project. If we sort points by the "Largest Residual" value, points with largest value needs to be reviewed. Largest residual value is the difference between the position of the marked point, and the estimated position of that point calculated by Photomodeler.

At some point we found that our model is quite satisfying for our purpose (Figure 11). This model was exported in two files: .OBJ file that contains the stecak's geometry information, and .JPEG file with the stecak's texture information.

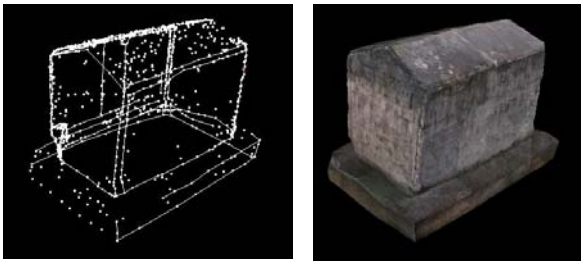


Figure 11: The final model of stecak in Photomodeler's 3D viewer

4.3 Improving the model in 3ds max

If we take a look at the previously created model in Photomodeler, we can see that its surfaces are mainly flat, and even if textures are fitting, we cannot see any relief of its surfaces.

In this paper, we introduce a new technique of improving models created in Photomodeler by photogrammetry technique using post-modeling in 3ds max software. We will modify its surfaces to obtain more relief.

The model is exported from Photomodeler as 3D Studio file (.3ds) with JPEG texture, and then imported in 3ds max. After importing, the model is converted to "Editable Poly", and we have applied three modifiers to it: Subdivide, Displace, MeshSmooth.

The first modifier we have applied to the model is Subdivide modifier. The main purpose of this modifier is to divide the object's surfaces in smaller parts, creating more faces. This modifier is used to prepare the model for the next modifier, and also for model optimization (Figure 12). We can optimize the model by selecting "Size" parameter value in Subdivide modifier.

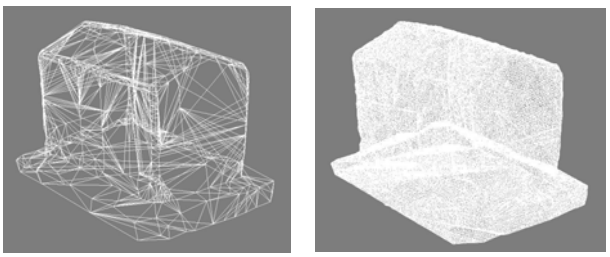


Figure 12: Model of the stecak before application of Subdivide modifier (left), and after

Subdivide modifier is used to prepare the model for Displace modifier. Purpose of using Displace modifier is to create relief on a totally flat surface based on the information from the texture map. Darker places in the textures will be pushed into the model, and lighter places will be pulled out of the model.

In our example, this technique of estimating relief proved as not so accurate, but it makes the model more realistic (Figure 13). We applied this modifier in the same amount on the whole object, but for getting better results, we could apply various amounts of this modifier to various parts of the object in different ways.

The last modifier applied on the model is MeshSmooth. This modifier smooths coarse, fragmented

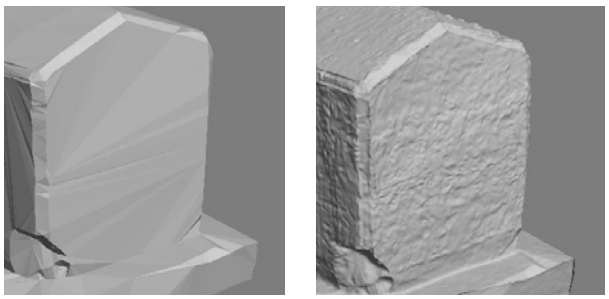


Figure 13: Model of the stecak before and after the application of Displace modifier

surfaces, and makes model surfaces to look more fluent (Figure 14).

After applying all mentioned modifiers, the model is exported as .OBJ file with additionally exported texture .JPEG file, and also in .WRL format, for usage on the web.

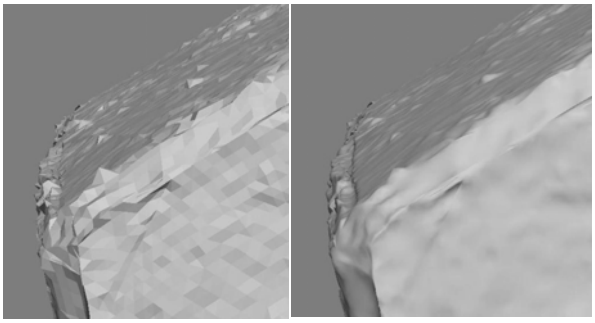


Figure 14: MeshSmooth modifier: The model before and after the application

We used the obtained model (Figure 15) as a part of our project "Digital Catalogue of Stecaks" [6].

Figure 15: The final model of stecak optimized for web

This project is a virtual museum of stecci from the collection of the National Museum of Bosnia and Herzegovina. The models created using photogrammetry combined with post-modeling work well considering all limitations of this online application.

5 Comparison of procedures

Laser scanning technique (LS) and photogrammetry technique combined with post-modeling (PP) are two completely different approaches for creating 3D models with high levels of realism. Both of them have some advantages and drawbacks which we will discuss in order to provide some recommendations for future projects of this kind. Results of this comparison based on our experience are presented in the following tables and figures.

We tested both techniques and compared their features. Results of this comparison are presented in Table 1.

Comparison of two techniques	LS technique	PP technique
Budget	High budget, expensive laser scanner	Low budget, cheap digital cameras
Geometry	High level of details	Low level of details
Textures	Problem with getting textures by night. Used only geometry, with radiance effects on the model's surfaces	Photo textures with high level of details
Operations taken automatically	The object's meshes generated using scanner	Automatic processing of marked data in Photomodeler software
Operations taken manually	Laser scanning. Editing and improving the model's meshes in Maya and Stitcher	Taking photos on site. Marking and referencing elements in Photomodeler
Accuracy level of model's geometry	High level of geometry details	Satisfying level of accuracy model in Photomodeler. Low level of estimated surface geometry details in 3ds Max
Optimizing	Not easy to optimize. For best results, hard work in manual optimizing is required	Easy to optimize. Models created in Photomodeler have very light geometry. Improved models in 3ds Max can be optimized easily in this software
Level of achieved realism	Geometry by itself offers good level of realism. In addition with good textures, the result is excellent	Very good level of photorealism. The model looks very natural to human eye
Human efforts	Overall small amount of included human efforts. If model's meshes have to be manually improved, human efforts can be much higher	Manual modeling requires a lot of human efforts in Photomodeler. Efforts in 3ds Max are usually low, but can be higher if high detailed mesh is needed
Required knowledge level	Laser scanning is not difficult to learn. Operator need to have earlier knowledge of 3D modeling and editing in order to know how to use similar programs. Editing model in Maya, requires essential knowledge of this software	Photomodeler is much easier to learn than programs like Maya or 3ds Max. Essential knowledge of calibration, marking and referencing can be achieved very quickly
Required experience level	Laser scanning process on site can be learned in a few hours. Experience of working in 3D CAD applications is desirable. Experience in working with Maya is needed	Only experienced operator can achieve well and accurate results in Photomodeler. Later improving in 3ds Max requires some experience of working in 3ds Max or similar CAD software

Table 1: Comparison of LS and PP techniques by various factors

The model created using laser scanning is optimized in several ways. Results are presented in Table 2.

Model ID	Description	Vertices	Faces	.OBJ size (KB)
LS_1[1]	Non-optimized, 5% of the scanned quality	1.051.544	1.666.396	274.288
LS_2	Optimized using Meshlab	472.447	827.151	70.862
LS_3	Optimized for web using Meshlab	86.384	150.000	12.402

Table 2: The Models created using LS (LSx[1])

The model created by the Photomodeler and then improved using 3ds Max is optimized in several different ways. Achieved results are presented in Table 3.

Model ID	Description	3ds Max Subdivide modifier, "Size" parameter	Vertices	Faces	.OBJ size (KB)
P[2]	Model, Photomodeler	-	657	1.299	211
PP_1[3]	3ds Max optimized model	1,5	195.166	389.844	41.560
PP_2	3ds Max medium optimized model	1	440.466	880.140	95.770
PP_3	3ds Max maximum tested quality model	0,8	636.870	1.272.720	139.233
PP_4	3ds Max model optimized for web	4	37.838	75.456	7.826 (8.988 for .WRL)

Table 3: The model created using photogrammetry (P[2]) and the models created using photogrammetry combined with post-modeling (PP_1[3]-PP_4)

[1] LSx - The model x created using laser scanning
[2] P - The model created using Photomodeler only
[3] PPx - The model post-processed in 3ds Max

As we can see, the optimization level of the final model created by PP can be easily adjusted using the "Size" parameter of Subdivide modifier. Optimization dependency of the parameter "Size" from Subdivide and number of vertices, number of faces and model's size (KB) is presented in Figure 16.

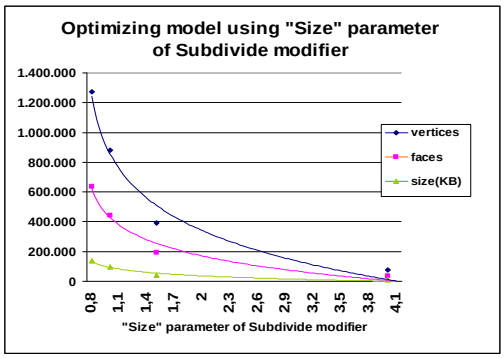


Figure 16: Optimization dependency of the parameter "Size" from Subdivide modifier and number of vertices, faces and size of model (KB)

Comparison of achieved photorealism in PP models is presented in Figure 17. These models are improved in 3ds Max and have various levels of quality, achieved using "Size" parameter of Subdivide modifier. In this way, we created a high quality model, optimized model and model intended for web usage (Figure 17).

Using MeshLab tools [5] we created from the laser scan the models with various levels of quality, Figure 18. Which quality of the model will be used depends of the project's requirements.

Figure 17: Comparison of models created by PP using Subdivide modifier: 1) created by Photomodeler (without post-modeling); 2) optimized with Subdivide "Size" parameter 1.5 (PP_1); 3) maximum tested quality - Subdivide "Size" parameter 0.8 (PP_3); 4) optimized for web - Subdivide "Size" parameter 4 (PP_4)

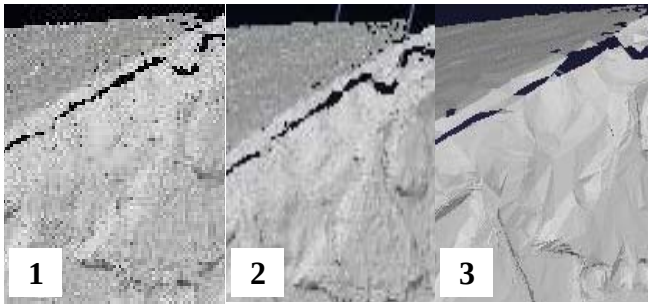


Figure 18: Comparison of various optimized LS models using MeshLab: 1) non-optimized model (LS_1); 2) optimized model (LS_2); 3) optimized model for web purpose (LS_3)

In our project we achieved various levels of geometry details in both techniques. Comparison of the highest and the lowest tested geometry quality is presented in Figure 19.

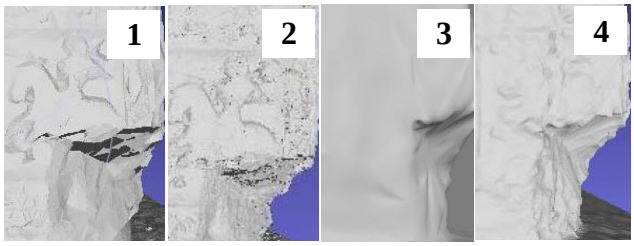


Figure 19: Geometry quality comparison: 1) optimized LS model for web purpose (LS_3); 2) non-optimized LS model (LS_1); 3) optimized PP model for web purpose (PP_4); 4) maximum quality PP model (PP_3)

5.1 Laser scanning advantages

The most important advantage of using LS is its accuracy. LS provided one millimeter accuracy in our example, and this should be more than enough for most of the applications. LS can be time saving. We can automatically generate 3D models using LS, and then optimize them using the software for automatic optimization in order to get good results. If we can extract textures from objects using this technology, then we can create models with high level of details, high accuracy, models that look very real.

5.2 Laser scanning drawbacks

LS is a very expensive technology. The raw scanned models usually have a lot of small mistakes that should be manually removed in order to achieve the high level of details. In our project we had problems with the laser scanner's daylight sensitivity, thus we scanned by night. Scanning by night caused another problem with textures, and we chose to exclude the scanned textures from the project. The optimization for web can be problematic because models created in this way consist of huge number of faces, and cannot be optimized for web without losing overall quality. Optimization can be very destructive on the models generated by LS. Still, we achieved very good results using MeshLab software tools [5].

5.3 Photogrammetry advantages

Photogrammetry is much cheaper than LS technique. If the budget is our main parameter, we should definitely choose photogrammetry. With this technique, we can also achieve very realistic models, especially if using photogrammetry in combination with manual post-modeling.

5.4 Photogrammetry drawbacks

This technique is time-consuming. It requires well experienced operator in order to achieve good results. If

we need high level of accuracy, then we should choose laser scanning. It is not completely impossible to improve accuracy of the model created by this technique, but it could acquire a lot of additional time to achieve that goal. This is still mostly manual technique that requires a lot of user interaction, but there are a lot of research efforts for achieving higher level of automation in photogrammetry [8] [10]. We can say that fully automated modeling using photogrammetry is still an ongoing research topic in this area of 3D modeling [9].

There are some common drawbacks in both approaches. Some objects cannot be captured with either of two techniques, for example objects made of glass, transparent, polished or mirrored and shiny materials [4]. Research is in progress to find solutions for these problems. Also, neither of these techniques is fully automated yet, in a way that manual editing is not needed at all.

6 Conclusion

In this paper we presented the procedures of laser scanning and photogrammetry combined with manual post-modeling applied for digitization of The Stecak from Donja Zgosca. The obtained results are used for comparison of these two techniques. Different ways of optimizing final models are also presented.

We introduced the concept of photogrammetry combined with manual post-modeling by improving our object in 3ds Max. This way we obtained more realistic surface details which were combined with high quality textures to achieve the satisfying quality of the model.

If we need to create a final model of the object with high level of realism, and we have a low budget, we should use some of photogrammetry based techniques. If we have requirements for creating a model with high level of accuracy, then we should use laser scanning technique which gives us more precise results. We can also use a combination of these techniques, photogrammetry for getting overall object geometry with realistic textures, and laser scanning technique for precise and accurate information of object's surface. The results obtained in this project can help us to decide which technique to use in our future projects.

References

[1] Selma Rizvic, Aida Sadzak, Emir Buza, Alan Chalmers, *Virtual reconstruction and digitalization of cultural heritage sites in Bosnia and Herzegovina*, Review of the National Center for Digitization, Faculty of Mathematics, Belgrade, Issue: 15/2009, pg 64-72, ISSN: 1820-0109

[2] S. Rizvic, A. Sadzak, Z. Avdagic, A. Chalmers, *Maya Sun Simulation of Bosnian Gravestone Virtual Model*, EuroGraphics Italian Chapter, Catania 2006

[3] Selma Rizvić, Aida Sadžak, Zikrija Avdagić, Alan Chalmers, *The Techniques of Virtual 3D Reconstruction of Heritage Sites in Bosnia and Herzegovina*, Sarajevo, ICAT05

[4] Paolo Cignoni and Roberto Scopigno, *Sampled 3D Models for CH Applications: A Viable and Enabling New Medium or Just a Technological Exercise?*, ACM Journal on Computing and Cultural Heritage, Vol. 1, No. 1, Article 2, Publication date: June 2008

[5] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, Guido Ranzuglia, *MeshLab: an Open-Source Mesh Processing Tool*, Sixth Eurographics Italian Chapter Conference, page 129-136, 2008

[6] Digital Catalogue of Stecaks, http://h.etf.unsa.ba/dig-katalog-stecaka/

[7] Virtual Sarajevo, www.virtualnosarajevo.com.ba

[8] Clive Fraser, Simon Cronk, Ida Jazayeri, Christos Stamatopoulos, *Automation in Close-Range Photogrammetry*, http://www.eng.unimelb.edu.au/research/themes/projects/sustainable_systems/automation_in.html

[9] Wolfgang Förstner, *Real-Time Photogrammetry*, Photogrammetric Week 05, 2005

[10] Claus Brenner, *City Models – Automation in Research and Practice*, Photogrammetric Week 01, 2001

[11] Simon Ratcliffe and Andrew Myers, *Laser Scanning in the Open Pit Mining Environment A Comparison with Photogrammetry*, I-SiTE Product Development White Paper, July 1, 2006

[12] P. Grussenmeyer, T. Landes, T. Voegtle, K. Ringle, *Comparison Methods of Terrestrial Laser Scanning, Photogrammetry and Tacheometry Data for Recording of Cultural Heritage Buildings*, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B5. Beijing 2008

[13] Alberto Guarnieri, Antonio Vettore, Fabio Remondino, *Photogrammetry and Ground-based Laser Scanning: Assessment of Metric Accuracy of the 3D Model of Pozzoveggiani Church*, FIG Working Week 2004, Athens, Greece, May 22-27, 2004

Fine Image Resampling Algorithm

Bronislav Přibyl*
Supervised by: Pavel Zemčík†

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

Abstract

This paper introduces a fine image resampling algorithm intended for corrections of image distortions caused by lenses or similar devices. The algorithm is designed for correction of small distortions in terms of pixel displacement but with high subpixel precision. The geometrical description of the correction is through bilinear interpolation within each node of a sparse square or rectangular mesh. The paper describes the algorithms itself, its features, implementation issues and data formats. Specifically discussed are the issues connected with programmable hardware (FPGA) implementation.

Keywords: Image resampling, Subpixel resampling, Lens distortion correction, FIR filter, Bilinear interpolation

1 Introduction

Image processing is one of the fields of computer science and applications that is developing very fast. The object of image processing is, of course, an image. Vast majority of image processing methods assumes that the image is a 2D signal represented through samples organized in a regular square or rectangular raster [2]. While the contemporary image acquisition devices and methods acquire images that relatively well fulfill the above assumption, in most cases, the images suffer from small geometrical imperfections caused e.g. by lenses (pincushion distortion, barrel distortion) used with the cameras that acquire the images.

The geometrical imperfections are in some cases not critical; however, many applications of image processing exist that suffer from the imperfections and where it is desirable to correct them. While the geometrical correction – calculation of new sample positions in the image – is relatively straightforward and can be e.g. performed through bilinear interpolation within square or rectangular mesh, the problem remains how to get the new samples values so that the signal properties of the image remain as much preserved as possible. Unfortunately, the nearest neighbor method, which completely destroys the image signal properties, and bilinear or bicubic interpolation [3] which can be better but by far is not ideal, are traditionally used for this purpose (see figure 1 for illustration). The main reason is that while the algorithms to preserve good signal properties, namely frequency spectrum, are known, they are often considered prohibitively computationally expensive. This paper proposes a method that is far better from the point of view of signal properties than the bilinear or bicubic interpolation while still preserves relatively low computational requirements. The limitation of the proposed method, however, is that it is limited to the cases where the distortions do not involve significant angular or scale changes – the method merely assumes only local subpixel shift limited to several pixels displacement [2, 3].

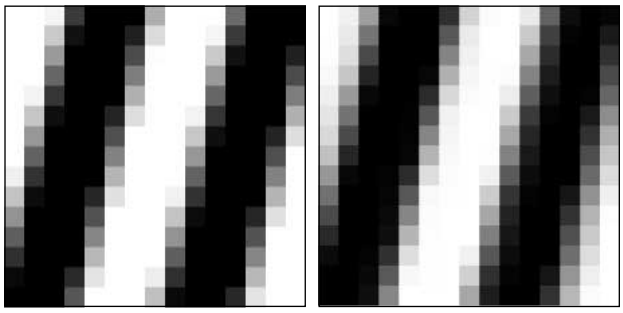


Figure 1: (**a**) Original pattern with 4 px thick synthetic lines 15 × magnified. (**b**) The same pattern resampled at scale 1.05 using bilinear interpolation. Note that edges of lines in the resampled image are significantly blurred.

2 Image Resampling

General image resampling problem is relatively straightforward mathematically – it is merely a problem of proper reconstruction of signal values in 2D space and proper application of sampling theorem. However, the efficient implementation of such resampling is still quite open problem. In our approach, we limit the general problem to

*xpriby12@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

resampling in order to correct geometrical imperfections only. This limitation has the following implications:

- The displacement of pixel location of the original and resampled images is only units of pixels,
- no angular distortion is expected, and
- no scaling is expected.

The general approach for resampling in our case is to scan the output image raster pixel by pixel (sample by sample) and reconstruct the values from the original raster based on knowledge of the pixel displacement. Taking into account the above limitations, it is known that the sampling theorem cannot be violated and also it can be assumed that the function is separable:

$$r_{x,y} = f\left(o, d(x,y)\right) = f''\left(f'\left(o, d(x,y)\right), d(x,y)\right), \quad (1)$$

where r is the resampled image,
o is the original image,
d is the displacement function,
f is a resampling function,
and f' and f'' are the partial reconstruction functions after separation.

In our case, the functions f' and f'' are implemented through a bank of FIR[1] filters [4] indexed by subpixel location of the pixel. Moreover, the sampling is the same in both directions, so f' and f'' are implemented using the same FIR bank.

The above solution with FIR filters was chosen as it has well defined features and as it is quite flexible in terms of exchangeability of the filtering function.

3 Proposed Resampling

The proposed approach to resampling relies on the separability; however, in addition to the generally used approach, the separability is applied to both the resampling function itself and the geometrical distortion calculation so that the distortion calculation is separated in vertical and horizontal directions:

$$r_{x,y} = f''\left(f'\left(o, d_x(x,y)\right), d_y(x,y)\right), \quad (2)$$

where r is the resampled image,
o is the original image,
d_x and d_y are the displacement functions,
and f' and f'' are the partial reconstruction functions after separation.

[1] Abbreviation from Finite Impulse Response.

The resampling function itself is assumed to be some suitable filter function and in the presented approach it is implemented through a bank of FIR filters. The bank of FIR filters is indexed through a subpixel position of the desired sample in the raster (see equation 3). The reason is that the FIR coefficients are dependent on the subpixel position of the desired sample location. Of course, the size of FIR filters is limited. The filters in the bank can be e.g. Lanczos filters [7] for optimal exploitation of the bandwidth of the image signal given the size of the filter, or other filter design to achieve the desired image signal properties. The described approach is, in fact, not dependent on it.

$$r_{x,y} = FIR_{fp(x)}\left(FIR_{fp(y)}\left(o, ip\left(d_y(x,y)\right)\right), ip\left(d_x(x,y)\right)\right), \quad (3)$$

where r is the resampled image,
o is the original image,
d_x and d_y are the displacement functions,
FIR_t is the function of the bank for position t,
fp is the fractional part of a numerical value,
and ip is the integer part of a numerical value.

The distortion to be corrected can be described in different ways, e.g. analytically or by a offset texture. Distortion description acquisition is not subject of this work but it has been studied e.g. in [5, 6, 8]. In our approach the distortion is described with a sparse rectangular mesh with displacement specified for each node of the mesh. In fact, the mesh can be seen as a offset texture. While the displacement in each node (corner of the rectangles) of that mesh is known, the displacement inside the rectangles is computed via bilinear interpolation. Size of the rectangles depends on application and local change of the displacement – the smaller size of the rectangle, the more precise approximation of the distortion but more memory for the distortion description needed.

Distortion inside each rectangle is described by means of the following four precalculated coefficients:

- D0 – displacement of top left pixel in the rectangle.
- DC0 – difference of displacements between adjacent pixels in 1st row of the rectangle.
- DR – difference of displacements between 1st pixels of adjacent rows in the rectangle.
- DDC – change in difference of displacements between pixels of adjacent rows in the rectangle, that means $DC_{n+1} - DC_n$.

For more detailed description see figure 2. Note, please, that the displacement calculation can be subdivided into independent calculation of vertical and horizontal displacements.

Displacement calculation executed by the resampling algorithm in each rectangle of the mesh can bee seen in

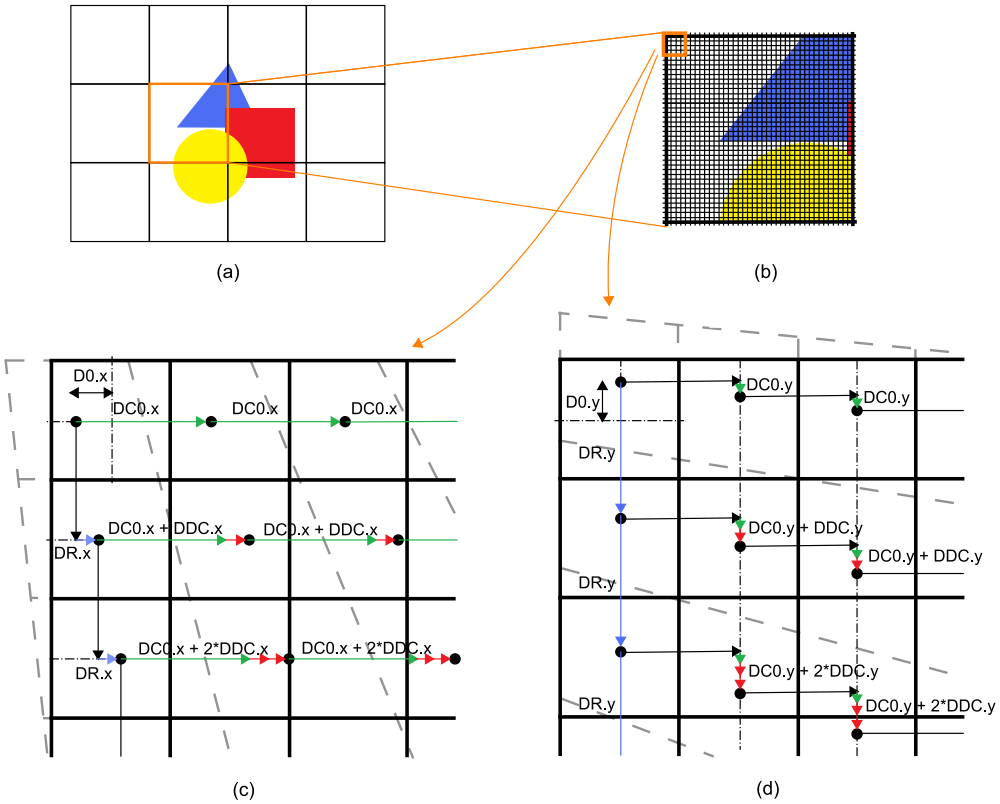


Figure 2: Displacement interpolation inside rectangles. (**a**) Output image is covered with rectangular mesh, source image distortion is defined within each mesh node. (**b**) Rectangular area of the output image with individual pixels visible. (**c**, **d**) Graphical representation of precalculated coefficients describing the source image distortion for horizontal and vertical resampling respectively. Pixels of distorted source image are plotted with gray dashed lines, pixels of output image are plotted with black solid lines. Big black dots are centers of source image pixels. Meaning of precalculated coefficients is marked with coloured vectors.

algorithm 1. The input of the algorithm is the original image, distortion description (through the above mentioned coefficients), and FIR filter; the output is the resampled pixels within the given rectangle. Note, please, that two instances of the algorithm are being used, one for vertical and one for horizontal displacement and filtering.

```
1:  var DoR, DC, D;
2:  DoR = D0;
3:  DC = DC0;
4:  for all rows in rectangle do
5:      D = DoR;
6:      for all pixels in row do
7:          Output FIR[fp(D)](O,ip(D));
8:          D += DC;
9:      end for
10:     DoR += DR;
11:     DC += DDC;
12: end for
```

Algorithm 1: Displacement calculation.

Time complexity class of the displacement calculation algorithm is $O(n^2)$ because displacement for each pixel of the output image needs to be computed directly. Cost of displacement calculation for a pixel is just cost of one addition operation (algorithm 1, line 8) plus cost of FIR filter response computation (algorithm 1, line 7). When the proposed FIR filtering is used, the time complexity class does not change and remains $O(n^2)$. This fact is, however, not important as the size of the image is fixed anyway and the proposed approach significantly reduces the computational cost.

As it can be seen from the pseudocode, three variables are needed in the displacement calculation algorithm. Their meaning is as following:

- DoR – displacement of 1st pixel in a row.
- DC – difference of pixel displacements.
- D – displacement of current pixel.

As shown in the algorithm, the displacement is subdivided into integer and fractional parts. The integer part is used to determine the pixel placement of the filter while the fractional part is used to determine the set of coefficients within the filter bank. When the number of filters in the bank is N (e.g. 16), the fractional part is multiplied by

N and then rounded to nearest integer. Then it is used for filter bank index.

4 FPGA Implementation

Two implementations of proposed approach have been realized so far; On CPU[2] using C[3] language and in FPGA[4] [1] using VHDL[5] language. The former one was originally intended as support and template for implementation of the latter one, so it does not fully exploit potential of the CPU. However it closely simulates processes running in programmable hardware with bit-precision, so outputs of both implementations are identical.

An FPGA platform has been chosen as it offers high degree of parallelism, which is exploitable e.g. in convolution-like operations. In our case it has been employed together with pipelining[6] to accelerate computation of the FIR filter response.

When implemented on CPU, time complexity class of filtering operation is generally $O(n^2)$ (when using only row or columnar filter it is $O(n)$) as a function of the filter size. This is because all pixels in the filtered area have to be multiplied by their respective filter coefficients sequentially and also summed up sequentially. On the other hand FPGA offers the possibility to multiply all pixels with filter coefficients simultaneously. However impact of this can be eliminated by pipelining which causes that results of the filtering operations are available in each cycle, only with some delay.

The overall structure of the resampling system is based on the fact that in vast majority of applications, the image is acquired "line by line" "pixel by pixel" from top left corner to bottom right corner. Also, in most of the memory storage structures, the images are stored this way. Therefore, it can be assumed that the image is best processed in the same order and the dataflow is adjusted so.

The overall structure is illustrated in figure 3. First, the image is fed into a buffer that can hold several complete image lines. The buffer must be large enough to accomodate as many lines of the image as it corresponds to the maximal vertical displacement in both directions plus the size of vertical FIR filter. In this case, the height of the buffer should be at least 31 pixels (2×12 px maximal vertical displacement + 7 px FIR filter height).

Next, the data fed into the buffer is processed so that for each line of the output image and for each position of a pixel on that line, the corresponding vertical location in the buffer is found. Then columnar neighborhood of that location, which is of same size as the FIR filter, is fed into the vertical resampling unit (in our case 7 samples). Thanks to parallelism of FPGA, the buffer allows to access the neighborhood in one clock cycle.

Then, the resampling unit calculates on output value of the vertical part of the resampling process through FIR''_q, where q is the index of a set of coefficients in the FIR'' filter bank determined from the fractional part of the vertical position. This concludes the vertical resampling process.

After the vertical resampling, the horizontal resampling takes place. The horizontal resampling part can take advantage from the fact that the data used in it is image line data only – the result of the vertical resampling unit. As the sampling frequency of the resampled image is very similar to the original image, it can be additionally assumed that for one pixel produced by the vertical resampling unit, approximately one pixel of the output image will be produced by the horizontal resampling unit as well. This allows for usage of only a small buffer (units of samples) between the vertical and horizontal resampling. In fact, the data buffer can even be in a form of a small shift register where the shift factor is 0, 1, or 2 samples at a time.

When the location of the data in the horizontal direction is determined, the actual horizontal resampling takes place using similar mechanism to the vertical one – the FIR'_q filter is applied based on the subpixel position in horizontal direction, where q is the index of a set of coefficients in the FIR' filter bank.

The horizontal resampling unit directly produces the resampled image pixels that should be stored into the resampled image data structures and this concludes the processing of the complete resampling unit.

In both vertical and horizontal resampling units, the control is done based on the calculation of displacements of resampled pixels according to the method described in section 3.

An FPGA implementation of the resampling algorithm has been prepared as part of the experiments with the design. The dataflow in the resampling unit can be seen in figure 4. Functional blocks are associated in two groups – one group handles vertical resampling while the other handles horizontal resampling. Each group consists of a FIR module, a Displacement interpolation module and respective Resampling module.

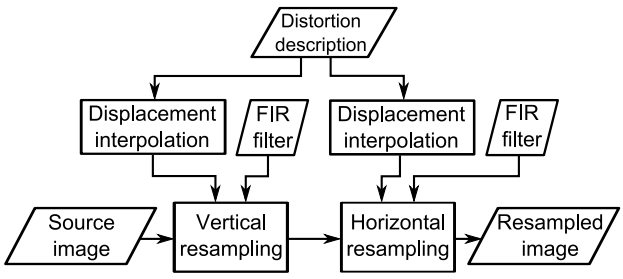


Figure 4: Dataflow of the resampling algorithm in FPGA.

Data formats used in the algorithm are the fixed deci-

[2] Abbreviation from Central Processing Unit – a processor.

[3] A general purpose programming language.

[4] Abbreviation from Field-programmable Gate Array – a type of integrated circuit which can be programmed after manufacturing.

[5] Abbreviation from Very-high-speed integrated circuit Hardware Description Language.

[6] Pipeline is a series of computational elements which work in parallel. Output of one element is input of the next one.

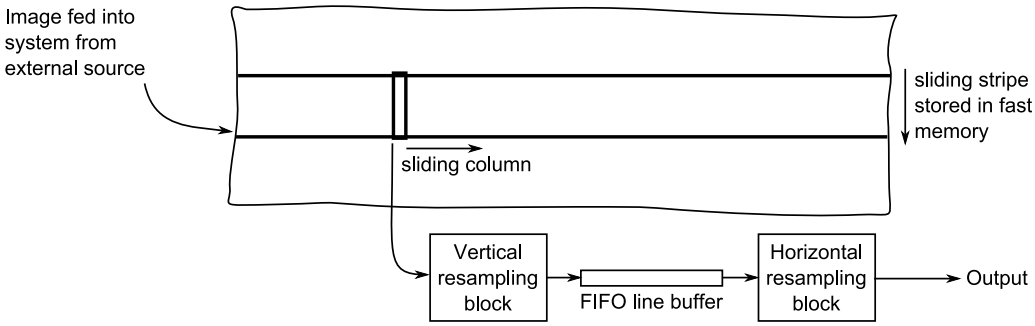


Figure 3: Overall processing of the image resampling.

mal point numbers in order to represent the data accurately enough while maintaining the design simple to enable its simple implementation.

The actual data formats used in the experiments are described below.

- Pixel data – 16 bit signed or unsigned. The pixel processing is assumed in 16 bit format in order to support the standard dynamic range of contemporary video cameras, which is 10 to 14 bits, plus an overhead for absorption of rounding errors of FIR.
- Coordinate – 12+4 bits unsigned. The subpixel resolution is assumed to be 16 subpixel positions which is in practical terms enough to avoid measurable adverse effects of granularity in subpixel position.
- Difference of coordinates – 2+8 bits signed. The difference of positions must be precise enough to represent the change of displacement.

Data formats of precalculated coefficients used as parameters of the displacement interpolation algorithm (see section 3) as well as data format of its output can be seen in table 1. Data formats specified in the table may seemingly not correspond to data formats described above. This is because all the coefficients and variables used in the interpolation algorithm do not represent coordinates of pixels. Instead of this, they represent displacements of output image pixels relative to input image pixels which have smaller range. Therefore upper bits of integer parts of the variables do not have to be used. On the other hand, more bits of fractional parts of the variables are utilized in order to absorb a cumulative error which emerges during execution of the interpolation algorithm.

The experimental design and synthesis of the resampling unit was performed for Xilinx[7] Virtex-II xc2v1000 FPGA device. XST[8] version H.38 was chosen for this task. As the unit is relatively generic, the following parameters were used: Image size 256×1024 px and square size 64 px which results in square mesh of 4×16 squares (and also displacement coefficient sets). Device utilization with configuration mentioned above is shown in table 2.

Items on chip	Used	Capacity	% capacity
Slices	3 947	5 120	77 %
Slice Flip Flops	2 112	10 240	21 %
4 input LUTs	3 103	10 240	30 %
BRAMs	20	40	50 %

Table 2: Exploitation of FPGA unit Virtex II-1000.

The device clock frequency is up to 105 MHz. While the resampling unit produces one output pixel per 2 clock cycles, the output resampling data rate for a single unit is up to 52.5 Mpixels per second. Thus the device is able to process 720p high definition video format (1280×720 px) at framerate 50 fps. This demonstrates the real-time potential of the design.

5 Results

The algorithm has been evaluated with images of artificial lines, photographic patterns acquired by camera and other images with results identical for standard CPU implementation and FPGA implementation. The resampling itself was performed with 7-sample Lanczos filter and the subpixel resolution was 16. These values are the limit values for the current implementation. These values can be seen as limits for efficient exploitation in real-life applications; However, they do not represent any limit for FPGA hardware.

Because no ground truth of an resampled image is available, we decided to use energy of power spectrum as a measure of error. The exact measure is ratio of energy of the resampled image and energy of the source image. An ideal algorithm would have the ratio 100 %.

The image of artificial lines (figure 5) was resampled at constant scale 1.05 in whole area using bilinear interpolation as well as our proposed algorithm. Power spectra of respective images are also shown. Bilinear interpolation algorithm proved to preserve 88.96 % of the spectrum energy, our proposed algorithm preserved 91.77 %.

[7] A company producing programmable logic devices, such as FPGAs.

[8] Abbreviation from Xilinx Synthesis Technology; A application for synthesizing device designs from hardware description language code.

Order of bit (2^n)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
D0 (signed)											s	•	•	•	•	•	,	•	•	•	•												
DC0 (signed)																	,		s	•	•	•	•	•	•	•	•						
DR (signed)																	,		s	•	•	•	•	•	•	•	•						
DDC (signed)																	,				s	•	•	•	•	•	•	•	•	•	•	•	•
Displacement (signed)											s	•	•	•	•	•	,	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 1: Precision of the interpolation algorithm coefficients and computed displacement. • denotes a used bit, "s" stands for sign, void bits are not used. Fixed decimal point is marked between bits nr. 0 and -1 with a comma.

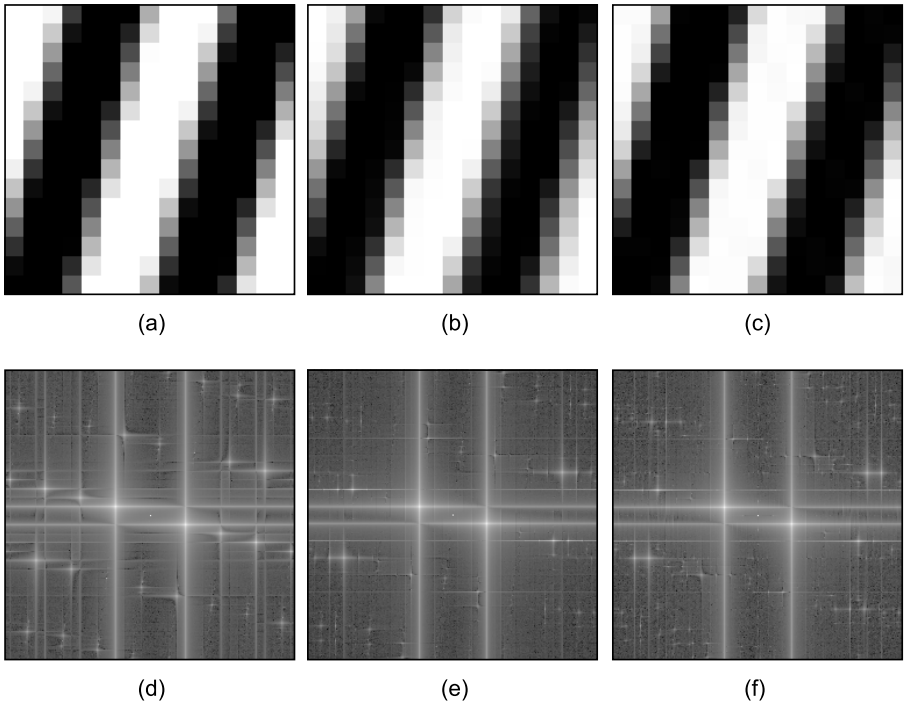


Figure 5: (**a**) Original pattern with 4 px thick synthetic lines 15 × magnified and (**d**) its power spectrum. (**b**) The same pattern resampled at scale 1.05 using bilinear interpolation and (**e**) its power spectrum. 88.96 % of energy of the power spectrum has been preserved. (**c**) The same pattern resampled at scale 1.05 using our proposed algorithm and (**f**) its power spectrum. 91.77 % of energy of the power spectrum has been preserved.

The image of photographic pattern (figure 6) was resampled simulating geometrical correction of barrel distortion using bilinear interpolation and our proposed algorithm too. Power spectra of respective images are shown as well because there are no observable differences in the output images. Bilinear interpolation preserved 90.32 % of the spectrum energy, our proposed algorithm preserved 98.45 %.

6 Conclusions

In this paper we described the accelerated fine image resampling approach based on a combination of a set of algorithms. Its intention is to correct geometrical image distortions caused by lenses or similar devices. Additionally, the implementation in software and FPGA is mentioned.

The presented approach and set of selected algorithms is based on approximation of the image distortion using a sparse rectangular mesh with bilinear interpolation of the positions within the nodes. The resampling itself is based on separable FIR filtering with a bank of filters indexed by a subpixel position.

As shown in the paper, the selected approach proved to be functional and lead into efficient implementation of resampling in both software and programmable hardware. The FPGA implementation is able to process up to 52.5 Mpixels per second which demonstrates the real-time potential.

It has also been shown that the selected approach gives better results than widely used bilinear interpolation algorithm. Additionally, we solved the problem of efficient implementation of a resampling algorithm, which properly reconstructs signal values in 2D space. However the general problem is limited by this approach to cases without angular distortion, scaling and significant pixel displace-

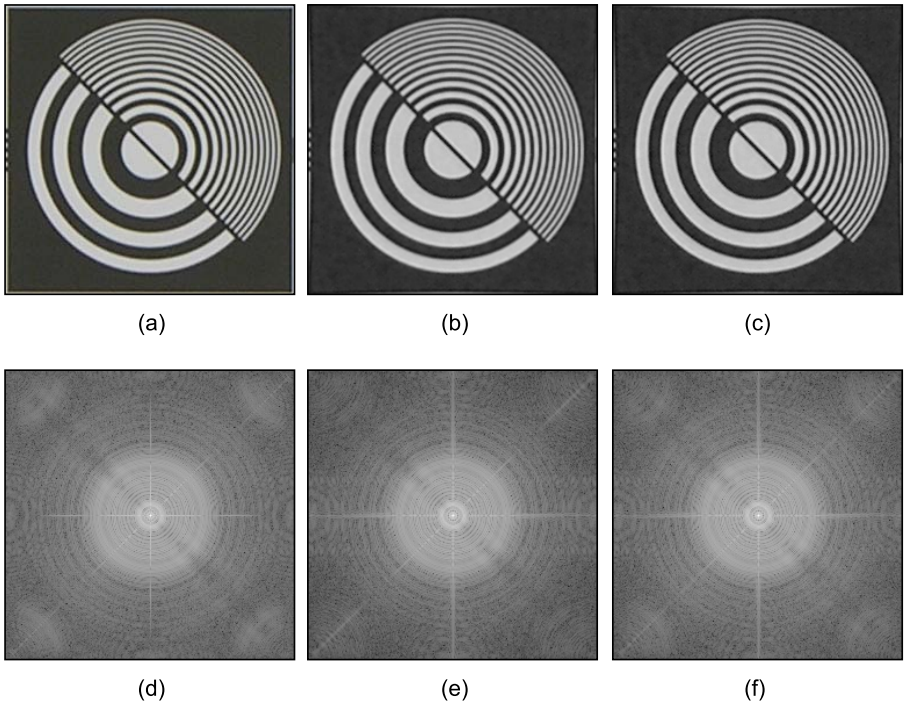


Figure 6: (**a**) Original image with a photographic pattern acquired by a camera and (**d**) its power spectrum. (**b**) The same image resampled using bilinear interpolation simulating barrel distortion correction and (**e**) its power spectrum. 90.32 % of energy of the power spectrum has been preserved. (**c**) The same image resampled using our proposed algorithm simulating barrel distortion correction and (**f**) its power spectrum. 98.45 % of energy of the power spectrum has been preserved.

ments.

Future work includes exploitation of the design in real-time applications of image processing. For example an efficient GPU[9] implementation will be considered. The work also includes further improvements of the structure and possible extension to 3D raster data.

7 Acknowledgements

This work was supported by the grant project of the Ministry of Education, Youth and Sports of CR, (MSMT 2B06052) project "BioMarker".

References

[1] S. Brown and J. Rose. FPGA and CPLD Architectures: A Tutorial. *IEEE Design and Test of Computers*, pages 42–57, 1996.

[2] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[3] A.C. Gallagher. Detection of Linear and Cubic Interpolation in JPEG Compressed Images. In *Proceedings of the 2nd Canadian conference on Computer and Robot Vision*, page 72. IEEE Computer Society, 2005.

[4] L.R. Rabiner and B. Gold. *Theory and application of digital signal processing*. Prentice Hall, 1975.

[5] C. Ricolfe-Viala and A.J. Sánchez-Salmerón. Robust metric calibration of non-linear camera lens distortion. *Pattern Recognition*, 43(4):1688–1699, 2010.

[6] GP Stein. Lens distortion calibration using point correspondences. In *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings.*, pages 602–608, 1997.

[7] T. Theußl, H. Hauser, and E. Gröller. Mastering windows: Improving reconstruction. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 101–108. ACM New York, NY, USA, 2000.

[8] T. Thormählen and H. Broszio. Automatic line-based estimation of radial lens distortion. *Integrated Computer-Aided Engineering*, 12(2):177–190, 2005.

[9] Abbreviation from Graphics Processing Unit. A specialized processor mainly used for accelerating computer graphics algorithms.

The Prototype Light Projection System for Cultural Heritage Reconstruction

Bartłomiej Specjalny*

Institute of Computer Graphics and Multimedia Systems
West Pomeranian University of Technology
Szczecin / Poland

Abstract

In this paper we present prototype of cultural heritage virtual reconstruction method by extracting three color pattern projected on object's surface. With rising accessibility to projecting devices and digital cameras new possibilities have risen. This paper focuses on automatizing active range scanner techniques with assumption to move pattern and not any of the scene elements including both object and devices. The goal was to acquire detailed point cloud for further reconstruction by determining shape of pattern lines. Results show that with the use of difference images, low-pass signal filters and line thinning methods a detailed high poly model can be obtained even without refining point cloud.

Keywords: virtual reconstruction of cultural heritage, 3D model acquisition, pattern projection, depth from photographs

1 Introduction

In the last years virtual reconstruction has been an essential problem in many practical applications. One of the applications is preserving and reconstructing cultural heritage in order to visualize catalogue or reconstruct actual historic objects of great value. The Wkryujście Altar presented in Figure 1, is one of the greatest cultural heritages both of the Polish and German nations in Western Pomeranian region. It is currently held in Szczecin National Museum, in Poland. Our goal is to create a technology to make a 3D geometrical reconstruction of the Altar. However, to protect the Altar against any damage all the test work were held on the test sculpture of the similar structure (see Figure 5) but relativly smaller.

The reconstruction is done by analyzing series of images. Each of images contains record of displayed light pattern. Each pattern consists of three lines with different colors. Colors used for the projection are main colors of the RGB color space both projected and recorded by camera. New Boolean images with extracted lines are created from three corresponding channels. Lines are filtered, posterized and thinned. The third dimension is extracted by lineform analysis. All gathered information are then extracted as the point cloud for further reconstruction.

In section 2 we provide information about the target object which is a piece of cultural heritage. Moreover, typical virtual reconstruction methods and previous work taken are described. Section 3 contains visualization concept and detailed information about implemented method. Section 4 documents results of the prototype reconstruction and presents encountered artifacts. Section 5 concludes the whole work and results. In this section we summarize final effect and propose future work.

In this paper we present prototype method for low cost virtual reconstruction. We adopt range scanning techniques to obtain high polygon surface of real object. We analyze set of images taken from one setup. We extract lines, and based on their shape we recreate point cloud.

2 Virtual reconstruction of cultural heritage

Target object - the Wkryujście Altar - is of the great cultural value despite the fact it was partially demolished. Because of its actual state it can not be rotated, moved or lifted. We would like to present its unique history and value.

2.1 The Wkryujscie Altar

The Wkryujście Altart is held in National Museum. It was founded about 1500 AC by prince Bogusław X for church in Wkryujście. Originally it consisted of eleven oak, polychrome reliefs and two figures of st. Paul and st. Peter. However, as the time passed the parts of Altar were damaged or lost (st. Paul figure is missing since 1900). It was during the Second World War when some of the reliefs were taken to Germany and four of them were confiscated by the Soviet Army. They were held in Kiel, Greifswald, Moscow and Riga. Most of the Altar parts were recovered by 2001 [2].

The Altar parts are currently in different conditions, due to the lack of conservation. This caused degradation of the wooden structure such as fractures and cracks, which gave

*bartlomiej.specjalny@gmail.com

Figure 1: The photograph of a sample plate of the Wkryujście Altar titled: Jesus Christ coronation with thorn.

the surface a specific structure. Mostly all of the paint fell of, and only small fragments were preserved. Also some of Altar parts were devastated by human hands: noses, hands and weapons fragments. The overall spatial shape of the subject is problematic because of complex surrounding fragments which are more likely sculptures attached to relief.

The object is unique and priceless testimony of medieval masters of wood-carving. That is why the National Museum wants to bring it back to public when most all of its parts are back. Even in its current shape it is worth seeing for medieval art connoisseurs. However, for ordinary man it looks like a regular wooden sculpture.

Due to the lack of information about the original shape of the Altar, the decision was made by the head restorer, to not change anything on the original Altar. This decision was made according to Venice Charter [1] and adequate Polish law. Instead, virtual reconstruction was held. Computer modelsnot only give museum or artists the ability to reconstruct paintings, but also missing parts of Altar without interfering with the original sculpture. Any attempts to recreate fragments of relief on the actual object would be considered devastating. This corresponds to "primum non nocere" (in English "First, do no harm") maxim that all works on the object have to preserve original substance as well as the object's material and immaterial value. This means that only the minimum interference in the object is allowed. It makes from conservation, restoration and renovation a multidisciplinary subject, since it allows to preserve objects of great historical and cultural value.

2.2 Reconstruction techniques

In our work we focus on the automated and accurate geometric reconstruction of the museum's objects like sculptures and monuments. We take into consideration that our target object will be Wkryujście Altar.

There are a few projects like The Digital Michelangelo Project [4] or Scanning Monticello Project [6] which focus on the single laser projection and triangulation method. This method gives very good results but due to size of installation and costs it could not be used for Wkryujście Altar. Furthermore, fast laser scanning methods with hand devices could be difficult to perform because of large scanned surface, despite its great accuracy.

There are the real time model acquisitioning methods permitting to rotate object kept in hand and continuously updating its geometry, also filling missing holes in the fly [9]. Our object however can not be moved because of its poor condition.

The first attempt of the reconstruction of the Altar involved using stereophotogrammetric method in which geometric properties of objects are determined from images [5]. Only low detailed model was obtained. It was caused by the fact that model was reconstructed from a whole piece of the Altar and most of details were lost. But other multiple view geometry methods [8] with the detailed images of the Altar could give good results.

However, we want to focus on the high detailed reconstruction that can be done without any special equipment with single scene setup. That is why we based on conventional range scanning methods [3]. Taking into consideration the features of the Altar (poor condition and limited possibilities of moving the sculptures) and low budget of the project we propose method that uses ordinary and commonly available projector and camera. We wanted also to focus more on the actual data that can be extracted from images than the information that could be established on the knowledge of the position of the camera or projector.

3 Model acquisition technique

The measurement setup consists of the projector 3M^{TM} MP76401, Canon^{TM} EOS 10D digital camera with Canon^{TM} 17-40 mm f/40 L USM lens and the laptop with MATLAB^{TM} software (Figure 2). The projector is placed in front of the object. The camera is set on the same horizontal plane as the projector and their view directions form 45° angle. This way, the calculated depth is twice longer than a pixel horizontal shift from its base position (we assumed that the distance from the camera to the object is much longer than the depths). Both the projector and the camera are connected to the laptop to display line pattern and automatically take pictures of the object. To generate line pattern we use MATLAB^{TM} PsychoToolbox 3 [10]. The camera is controlled by Canon^{TM} SDK 2.5.2 and Canon^{TM} Remote Capture 2.7 software.

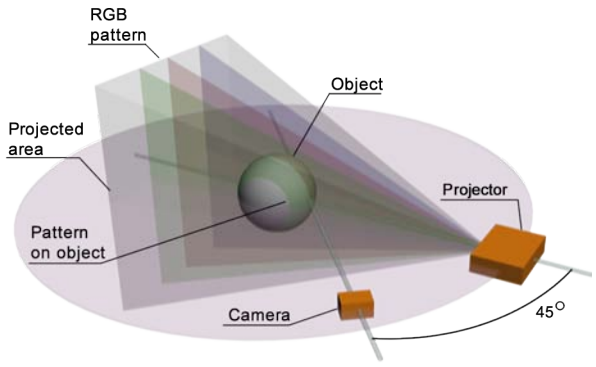


Figure 2: Model acquisition setup.

Before scanning, calibration takes place. Single dot is displayed in the middle to center the camera on the middle of projected area. Afterwards, we run line calibration to get indices of the two boundary lines enclosing the object. It allows to limit the number of required photographs and to estimate proper distance between the lines. Finally, a single shot of the object lighted by black image from the projector is taken (this image is called "base image Y"). This photograph will be used to reduce noise in the acquired data and extract color lines.

During scanning, the projector displays the vertical line pattern (Figure 4). The pattern is moved one pixel right after each photographs is taken (photograph is called "pattern image X"). The measurement is finished when the whole surface of the object is covered. The pattern consist of three vertical lines displayed simultaneously and set in constant range. This technique decreases by three times the number of required photographs without influencing the accuracy of the measurement.

The accuracy of reconstruction is limited by the projector resolution. It is desirable to project on the object as thin lines as possible. The lines should be sharp and the camera and the projector should be focused accurately.

After gathering all the input images, the workflow presented in the Figure 3 is executed. We compute the difference matrix XY for every pattern image X and base image Y. It helps to remove noise and highlights from the object. Then, difference matrices XYR, XYG and XYB are computed for each color channel. Additionally, a modifier can be used in the above equations to reduce exposition. We empirically found that this value should be set to 3, although, for less illuminated scenes it can be set to 1.

The low-pass Gaussian filter reduces noise and highlights. It also makes line smother and covers some of the missing parts. However, the filter kernel can not be too large because it could deform the lines. After Gaussian filtering, we posterize each channel to obtain the logical matrix containing information about lines (empirically determined thresholds are used). Since lines on logical matrix are a few pixels wide we reduce their thickness basing on the morphological operations algorithm [7].

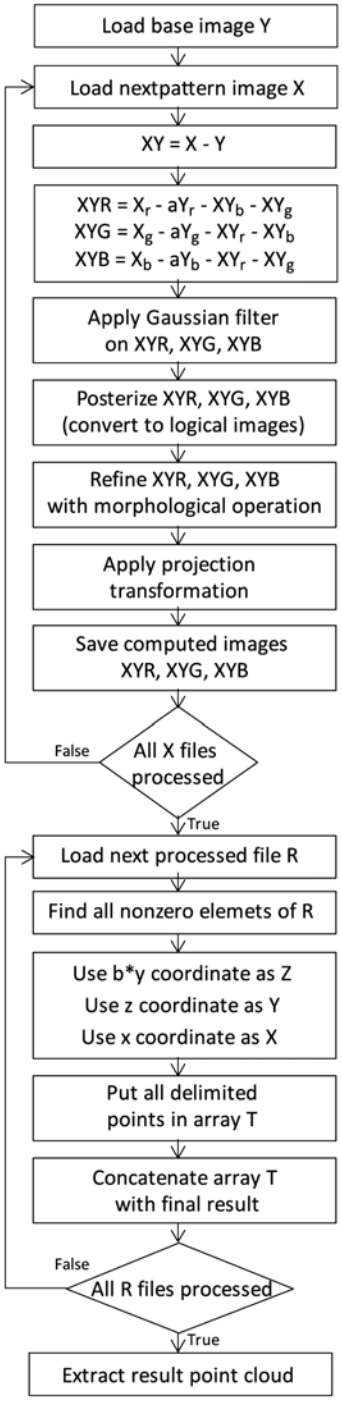


Figure 3: The flowchart of data processing, X,Y and XY represent all pixel from RGB image channels with values between 0 and 255, r,g and b stand for color channels index, XYR, XYB, XYG and R represent one channel images, additionally R is Boolean logical image, a is real modifier for image difference and $a > 1$, b is real modifier for horizontal distortion height correction and $b > 1$, T array contains all found vertexes that belong to a projected spline on the surface of the object.

Figure 4: The RGB light pattern projected on the object.

In the next step the projective transformation is conducted to reduce the perspective deformation of the object. Lines that are closer to camera are longer then the most distant ones. Their length should be reduced and the choice of what side to decrease/increase depends on which side of the projector the data is recorded.

Computed images XYR, XYG and XYB are saved maintaining number of line index. This way they are already sorted. After processing all files we reread them one by one as R matrix. Non-zero elements in R represent points in the point cloud. The coordinates of these points are computed basing on the position of the line and the pixel shift (see Figure 3). Height modifier b is applied to minimize flattening caused by taking picture from 45°. Extracted points are pushed into temporally array T. Before loading new sets of points T is concatenated with the final result array.

We export final point cloud to ASCII vertex file ($*.asc$). Mesh reconstruction is done in external software: Point Cloud 1.0 and Delaunay2_5D.

4 Results and discussion

In this section the results of the shape reconstruction of the object are presented. We tested our algorithm on the sculpture presented in Figure 5. It has similar structure to the Wkryujście Altar. The sculpture is also made of wood and has similar color as well as the surface texture.

The reconstructed mesh is presented in Figure 6. It consists of 403 317 triangles and was created basing on 203 672 points of the point cloud depicted in Figure 7. The points were gathered from 50 photographs of the light pattern and 1 base reference image.

Figure 7: The final raw point cloud acquired from a set of 50 RGB pattern photographs.

The shape of the object was preserved correctly with most of its details. The fidelity of reconstruction directly depends on the resolution of the projected pattern. For comparison, the mesh generated from lower number of input photographs (it means lower number of lines per the object surface) is presented in Figure 8. As it can be seen, less details were reconstructed in this mesh.

There are some artifacts in the final mesh. The main problem is a low scan resolution on surfaces parallel to the camera view direction (see the cheeks in Figure 6). This

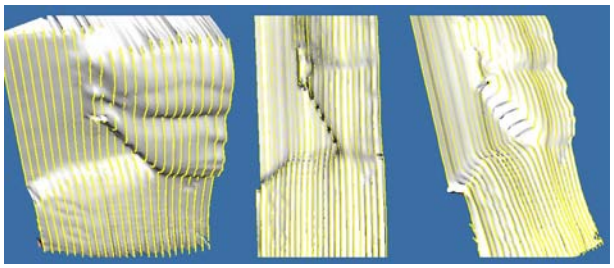


Figure 8: The mesh reconstructed with 8 RGB photographs. The yellow lines depict location of the projected lines.

Figure 5: The test object used to prototype the reconstruction technique. We can not use the Wkryujscie Altar because of its actual condition and prototype method that is still developed.

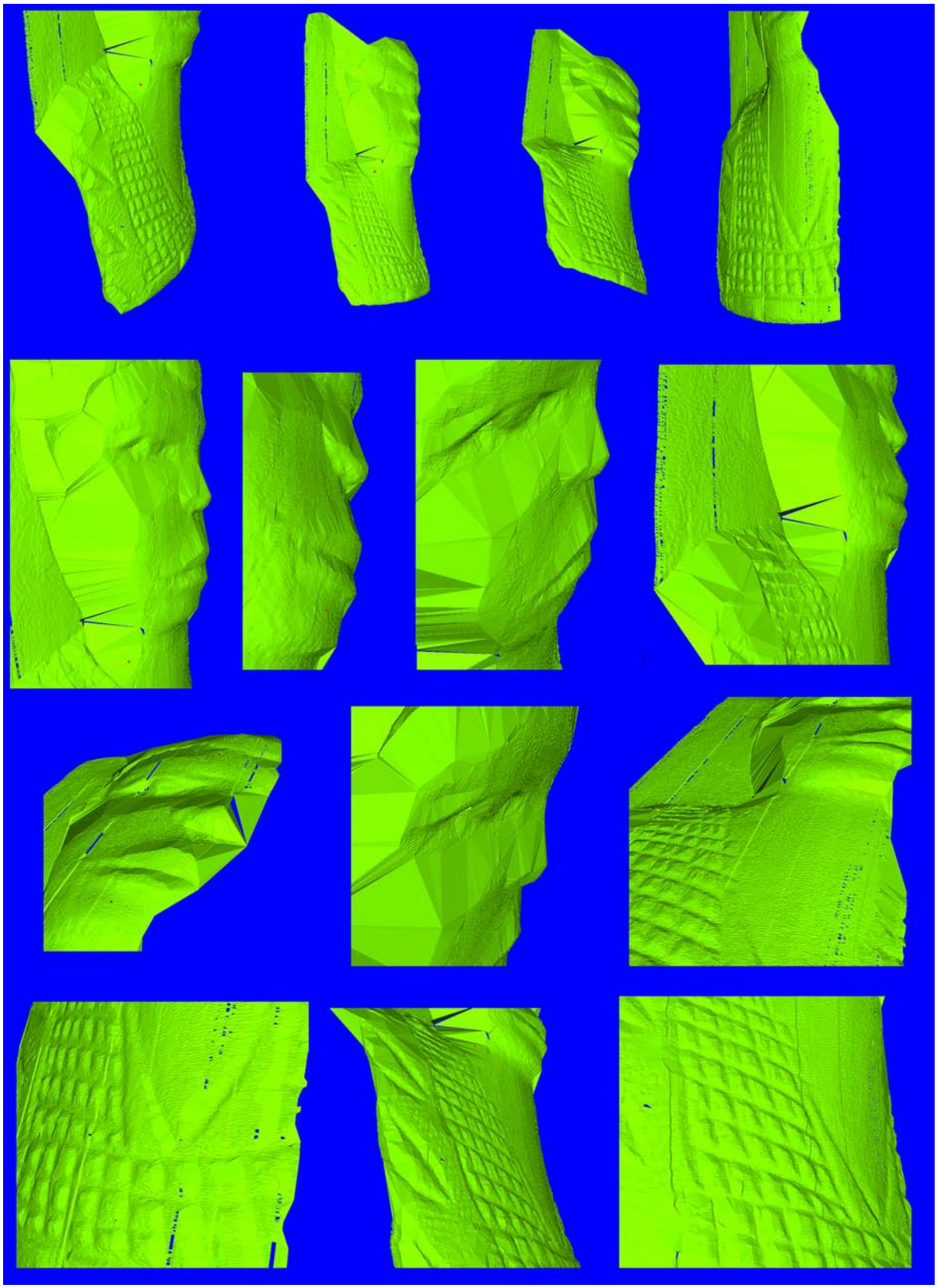


Figure 6: The final mesh reconstructed from the raw point cloud.

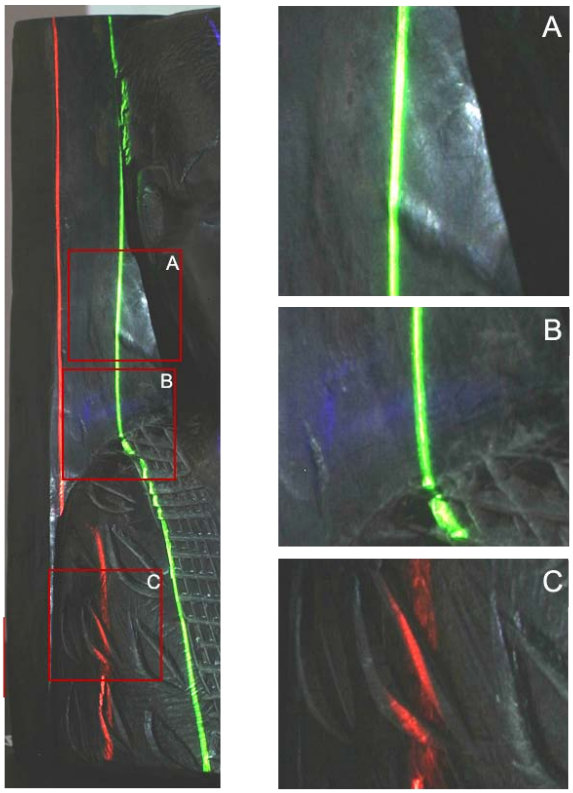


Figure 9: A sample close-ups of the light pattern projected on the object. A. The fluctuation of color caused by the highlight. B. The reflection of a blue line, this line is actually not visible in the presented image area. C. Red line splits into three separate lines.

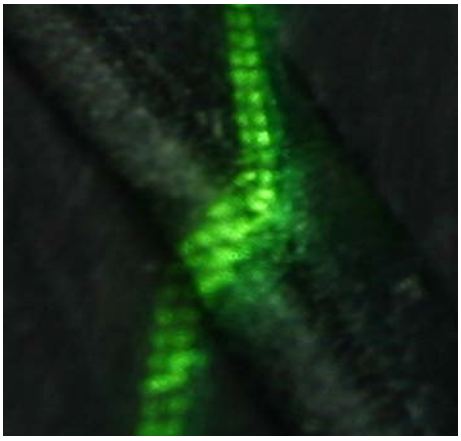


Figure 10: Magnified fragment of a green line. Its structure is distorted by the object surface. The visible green squares are effect of the projector resolution: every 1 pixel line consist of 2 projectors dots width. The projected lines has width from 1 to 3 millimeters.

drawback can be eliminated by mixing photographs taken from different directions. Some inaccuracies are caused by the material structure which splits a line into two or three (see Figure 9c). The lines have discontinuities and altering thickness (see Figures 9 and 10).

The results can be improved using advanced resampling and smoothing methods that would increase the fidelity of lines acquisition. What's more, more sophisticated the vertical perspective correction would correct the mesh proportions by making result model less flatten.

5 Conclusions and future work

In the paper we describe the prototype of the technique for the reconstrucion of object's shape. It is based on light pattern projection on the object surface. The photographs of the pattern deformed by the object irregularities are analyzed and virtual mesh of this object is reconstructed. We use inexpensive camera and projector but even with the use of this equipment high detailed elements can be correctly captured (see Figure 6). The missing regions can be filled by making another scan of the object from different camera position. We leave this issue for future work.

The presented technique is addressed for the Wkryujście Altar - the cultural heritage of the West Pomeranian Region. The shape and surface structure of the Altar is suitable for this type of scanning. After finishing the prototype phase we plan to make a scan of the whole Altar.

The accuracy of the reconstruction depends on the projector's resolution and its optical quality. We plan to use better projector for the actual scan. The using of the analog projector is also considered. Further improvement could be made by utilizing more sophisticated method of noise reduction and line acquisition.

Acknowledgements

I would like to thank Radosław Mantiuk for his continuous support during my work and presented paper. I appreciate all his advices, comments and patience. Additional thanks to Mariusz Borawski for his support with image analysis methods. I would also like to thank my friend Sławomir Mazgaj who was in the beginning working with me on this project.

Specially I would like to thank Marta Kotecka. I owe her my motivation for everything.

References

[1] Venice charter. Venice, 1964.

[2] Uwe Albrecht. *Po 60 latach znowu razem. . . , Zmienne koleje losu ołtarza z Wkryujścia (in Polish)*. 2009.

[3] Brian Curless. From range scans to 3d models. In *ACM SIGGRAPH Computer Graphics*, 1999.

[4] Brian Curless Szymon Rusinkiewicz David Koller Lucas Pereira Matt Ginzton Sean Anderson James Davis Jeremy Ginsberg Jonathan Shade Duane Fulk Marc Levoy, Kari Pulli. The digital michelangelo project 3d scanning of large statues. In *ACM SIGGRAPH Computer Graphics*, 2000.

[5] Krzysztof Miler. Rekonstrukcja wirtualnych modeli obiektów muzealnych (in polish). Master's thesis, West Pomeranian University of Technology, 2009.

[6] Kok-Lim Low John Thomas Kurtis Keller Lars Nyland David Luebke Anselmo Lastra Nathaniel Williams, Chad Hantak. Monticello through the window. In *4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*, 2003.

[7] Steven Eddins Rafael Gonzlez, Richard Eugene Woods. *Digital Image processing using MATLAB*. 2009.

[8] Andrew Zisserman Richard Hartley. *Multiple View Geometry in computer vision*. 2006.

[9] Marc Levoy Szymon Rusinkiewicz, Olaf Hall-Holt. Real-time 3d model acquisition. In *ACM SIGGRAPH Transactions on Graphics*, 2002.

[10] Psychtoolbox Wiki. http://psychtoolbox.org.

Automatic Image-Based 3D Head Modeling with a Parameterized Model Based on a Hierarchical Tree of Facial Features

Peter Kán*
Supervised by: Andrej Ferko†

Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava

Abstract

The automatic modeling of a 3D human head has challenged researchers in computer graphics for many years, because 3D head models are useful in several application areas to assist in different tasks. In this paper we present the novel system for automatic 3D head model creation from two images based on a parameterized head model with a hierarchical tree of facial features. The proposed system is divided into three parts. In the first one we use computer vision techniques like Skin-Tone based image segmentation or Haar Cascade Classifiers to detect head parameters for parameterized model update from frontal and profile head photograph. Then a 3D head model is reconstructed by updating a parameterized model with parameters detected from images and a texture is mapped. In the second part the reconstructed head model is rendered with real-time rendering techniques simulating skin illumination. In the third part the model is exported to the Collada format for further use in other applications. The system for automatic image-based 3D head modeling can be used e.g. in computer games development, movies, telecommunications, medicine, or security systems for human identification.

Keywords: 3D head modeling, face detection, facial features detection, face reconstruction, parameterized model, real-time rendering, Collada

1 Introduction

A 3D head model is an essential part of many computer graphic applications and its easy creation is an important task for developers. Manual 3D head model creation is very time consuming, especially when good precision is needed. The Automatic 3D head creation algorithm with few or no user interaction is a challenging task.

A parameterized head model [6] provides us possibility of creation many variable 3D heads and we can obtain a model representing concrete person if correct parameters and texture are used. In this work we created a new parameterized head model based on a hierarchical tree of facial features which enable application to easily estimate correct parameters for a specific human head. This model defines vertex weights for each facial feature, which gives us the information about transformations intensity in each vertex.

To gather concrete head parameters we use computer vision and image processing techniques like Haar cascade classifiers [23] and skin tone based image segmentation [14, 12, 17, 18]. Then we analyze detection results and estimate final head model parameters.

According to detected parameters from input images we calculate global and local transformations and apply them to create new model. Then we interpolate between old and changed model with parameters defined as vertex weights for each facial feature.

Our parameterized model consists of 32672 triangles to reach good reconstruction precision and to gives us high resolution head model. This model can be rendered with texture-space diffusion [11] technique even approximating subsurface scattering in interactive frame rates.

The paper is organized as follows. In section 2 previous works are discussed. In section 3 main parts of our modeling system are described, our novel parameterized head model based on hierarchical tree of facial features is discussed and rendering techniques simulating skin illumination are shown. In section 4 our reconstruction system results are shown and in section 5 we discuss conclusion and future work.

2 Related Work

The automatic head detection and reconstruction is an important task solved in research area for a long time. Many techniques were created to reach the best results. Head detection and reconstruction are in some approaches strongly related. The automatic head model creation techniques use head detection results and vice versa.

Reconstruction results within different approaches are variable in quality and computation time according to the purpose of use. Techniques used in telecommunications

*peterkan@peterkan.com
†ferko@sccg.sk

create a low-polygonal face or head model to achieve lower data transfer in videoconferences. For this purpose Mikael Rydfalk [22] created a parameterized face model named Candide in 1987. His model was a low-resolution adjustable model, which can be updated by several parameters. This model was later improved by Jörgen Ahlberg [6] to simplify animation and it was named Candide-3. An advantage of the Candide-3 model is fast reconstruction time, but disadvantage of this technique is low precision and few details.

Another approach generating high quality 3D head models, called a Morphable Model, was proposed by Volker Blanz and Thomas Vetter [8] to reach photorealistic model appearance and high precision in reconstruction. In this approach head model is represented as a multidimensional 3D morphing function based on the linear combination of large number of the 3D face scan. The main disadvantage of a Morphable Model is high computational time. This model was later updated by Yong-Li Hu et al. [15], who used the mesh resampling method to overcome the key problem of model construction, the pixel-to-pixel alignments of prototypic faces. When applying the method to several images of a person, the reconstructions reach almost the quality of laser scans.

Image-based head model reconstruction techniques are much more available and applicable. In some of them a head is reconstructed from one image like the AvMaker [16] program or reconstructed from several images. The special case of image-based head reconstruction is gathering 3D coordinates of points from two 2D images using stereo vision [20]. Advantages of image-based head model reconstruction techniques are generally fast computational time and low reconstruction cost. The main disadvantage is low reconstruction accuracy.

An Active Appearance Models [10] is an example of head model reconstruction technique based on predefined model and it is used mainly for head tracking in video. Liu et al. [21] use video sequences to generate 3D head models. A disadvantage of this approach is low acceptable head rotation angle.

In our solution image-based 3D head model reconstruction technique with predefined parameterized head model based on a hierarchical tree of facial features is used. This approach is capable of giving us the reconstructed 3D head model in a few seconds.

3 Virtual reconstruction of a human head

We created the Modeling system, which consists of the three main phases (Figure 1). In the first phase input images are loaded, head is detected in both images and textured 3D model is created. Then the reconstructed model is rendered in second phase with real-time rendering techniques, which simulate skin lighting. In the third phase the model can be exported to Collada format for further use in another application. The texture is also exported with a model given.

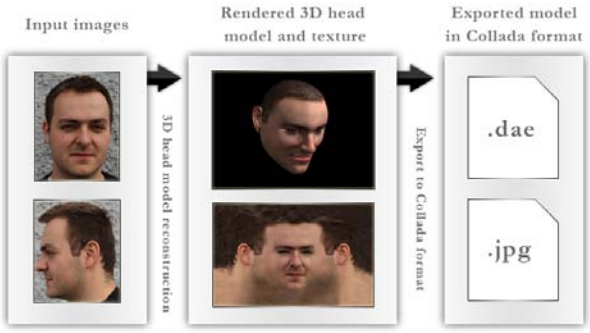


Figure 1: Three main phases of the modeling system

The modeling system detects facial features positions and head attributes automatically from input images and also allows user to adjust detected positions to enhance reconstructed model precision.

In our work a 3D head model is automatically reconstructed from two input images. For this purpose we created the new parameterized head model based on the hierarchical tree of facial features (Figure 4).

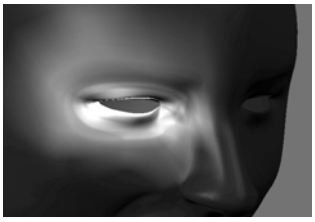


Figure 2: Example of vertex weights for the left eye. White color means weight value 1 and Black means 0.

All of these features have defined their own area of vertices. This assignment is done by vertex weights for each region as we can see in Figure 2. 3D positions of facial features are calculated from automatically detected 2D positions from input images. Facial features are organized in the hierarchical tree, where root node is the head center and child nodes are main facial features. Child nodes of the facial feature are its properties or other more detailed features. For example Left eyes nodes define left, right, top and bottom border coordinates in front image. The reconstruction system can easily adjust head parameters with the hierarchical tree of facial features, because in each facial feature position estimation all subnodes are precalculated to help find a real feature position. Another advantage of this approach is a possibility of simple user interaction and manual adjusting of the head model.

The reference mesh (Figure 3) of the parameterized model automatically controlled by the hierarchical tree contains 32672 triangles to achieve high resolution output

model with better precision and smoothness.

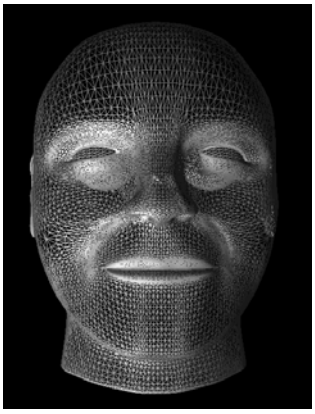


Figure 3: Reference mesh of parameterized head model

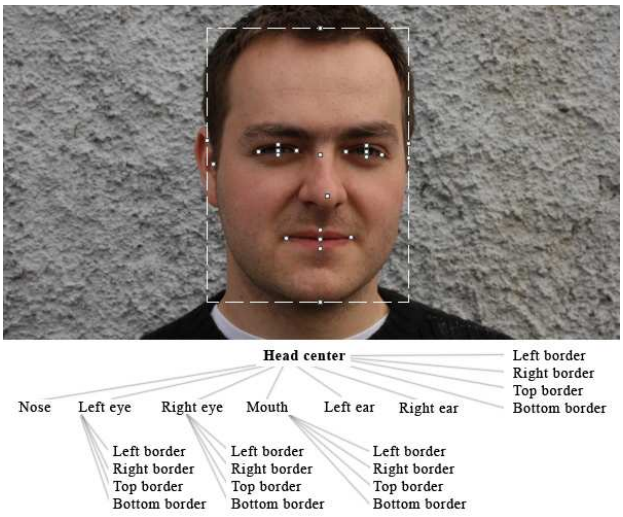


Figure 4: The hierarchical tree of facial features and theirs attributes. In bottom image are nodes in the tree and in top image is their visualization.

3.1 Head Model Reconstruction

Our automatic 3D head modeling algorithm with a parameterized model based on a hierarchical tree of facial features generates a three dimensional representation from two 2D images in two stages. In the first stage head position, facial features positions, and head attributes are detected from frontal and profile images using computer vision and image processing techniques. In the second stage is the parameterized model transformed to shape values detected from input images.

3.2 Face Recognition and Parameters Detection

Frontal and profile head photographs are expected as an input for the reconstruction system, therefore face and its properties detection should be performed separately for both images. To recognize the head and gather its properties from images two main techniques are used.

Haar Cascade Classifiers based on an extended set of Haar like features [19] are employed to detect face and facial features in the front image. For this purpose classifiers trained to detect Front head region [19], Eyes positions, Nose position [9] and Mouth position [9], and facial features proportions are used. The results of the face region and facial features detection can be seen in Figure 5.

To detect a head shape in the front image a Skin tone based image segmentation [14, 12, 17, 18] is used. The image encoding is converted into YCrCb color space and for all pixels Cr and Cb values are evaluated. If Cr and Cb coordinates of the pixel are in a skin tone area (Figure 6a), the pixel is segmented as skin-tone colored. By this technique all pixels are divided into skin tone pixels and non skin tone pixels as we can see in Figure 6. Then we can analyze skin tone colored pixels and acquire information about the head shape.

Figure 5: Face region and facial features detection by Haar Cascade Classifiers. Facial features are marked with a cross and face region with a rectangle.

Figure 6: Skin tone based segmentation. (a) Adjusted skin-tone area in CrCb color plane. Black pixels give us the information about Cr and Cb value considered as skin-colored. (b) Input image, and (c) segmentation result.

A color space selection is an important choice in Skin tone based image segmentation technique. We examined three color spaces Normalized RGB, Lab and YCrCb to select one, which will give us the best segmentation results. We chose chrominance plane from each color space to have the most chrominance information stored in two

selected channels. Skin tone areas in tested color spaces can be seen in Figure 7. Finally we chose YCrCb color space, because it gives us the most covered facial area by skin tone segmented pixels, and we manually adjusted skin tone area in CrCb plane (Figure 6a) to achieve better segmentation result. Skin tone area is stored in 255x255 binary image in our reconstruction system, because Cr and Cb values can reach maximum value 255.

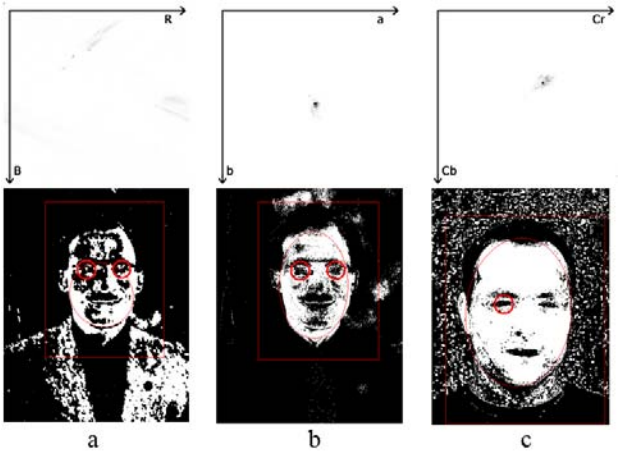


Figure 7: Skin tone areas (top) and segmentation result (down) in Normalized RGB (a) Lab (b) and YCrCb (c) color spaces. Skin tone areas were created by selecting skin areas in images from training set and adding skin pixel values into the skin tone area.

Figure 8: Profile face and facial features detection. From input image (a) the binary image is created by the skin tone based segmentation (b). The segmented image is filtered by a median filter and moving probes are sent from head center to each direction (c) to find borders. Then a head bounding box (d) is calculated. Finally a front head contour is obtained by sending moving probes from left border to right and gathered face curve is analyzed.

To detect parameters from the profile image (Figure 8) we use the skin tone based segmentation and then we apply a median filter to smooth the segmentation result and reduce noise. After median filtering we calculate head center by a linear interpolation of skin tone colored pixels' coordinates. We send moving probes from head center to each direction. These probes stop moving when they reach non skin tone colored pixel. After all probes stop, we calculate head bounding box by getting minimum and maximum x and y coordinate of probes. Then we obtain front head contour by sending moving probes from left to right border. Probes stop if they reach skin tone colored pixel. After all probes stop, we smooth out the result curve by a median filter and we can analyze it to obtain the information about tip of the nose position, eyes and mouth position. To analyze back head contour we send moving probes from right border to left.

As input images to parameters detection could be used photographs in an optional resolution and quality, but we recommend to use images with resolution 512x512 and higher, where more information about facial features is included and high resolution texture could be created.

3.3 3D Model Update

Updating model parameters is a key point in the 3D head reconstruction with the parameterized model. In this step new vertex coordinates are calculated to shape head from input images. Head color texture and normal map texture are created from input images and they are mapped to reconstructed model geometry to reach more realistic result in rendering.

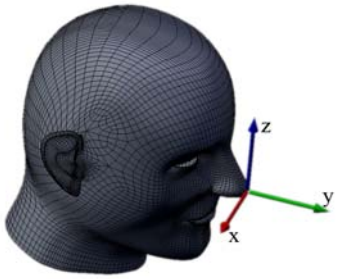


Figure 9: The parameterized head model with a reference coordinate system aligned to the tip of the nose.

All features in the parameterized model based on the hierarchical tree of facial features have defined vertex weights to transform correct vertex due to the features coordinates in the hierarchical tree calculated in the head recognition phase. These weights were precalculated in a training process (Figure 2). Each facial feature has also defined its own local coordinate system, which center is stored in our parameterized model and usually it is in the center of facial feature. Rotation of local feature coordinate system is same as in global coordinate system.

Firstly we need to define a model coordinate system origin to perform image space to model space position calculation. In our model a tip of the nose was selected as a reference point (Figure 9) of a 3D model coordinate system. The first step in updating model phase is global scale transformation application. We don't change head model width in global transformations. According to model width and head aspect ratio in images we calculate vertical head scale along z coordinate and depth head scale along y coordinate as

$$V_s = \frac{FH_{2D}W_{3D}}{FW_{2D}H_{3D}} \quad (1)$$

and

$$D_s = \frac{PW_{2D}H_{3D}}{PH_{2D}D_{3D}}, \quad (2)$$

where V_s is a vertical scale coefficient along z axis, D_s is a depth scale coefficient along y axis, FW_{2D} and FH_{2D} are head sizes in front image, PW_{2D} and PH_{2D} are head sizes in profile image and W_{3D} is 3D head model width, H_{3D} is 3D head model height and D_{3D} is 3D head model depth in model space. After calculating V_s and D_s we scale all vertex positions in the 3D head model with those coefficients. Global scale is performed in coordinate system with origin in head center to preserve head center position.

After performing the global scale we translate head model to have the tip of the nose in the origin (0,0,0). Then we can make local head transformations in local features coordinate systems on vertices with weights defined per each facial feature in the hierarchical tree. We calculate new features positions and sizes according to the location in front and profile image. Then we calculate a final vertex position by the linear interpolation between new features positions and their old locations.

$$Vertex_i = (1-p).oldVertex_i + p.newVertex_i \quad (3)$$

According to equation 3 we calculate the position of i^{th} vertex $Vertex_i$ from old i^{th} vertex position $oldVertex_i$ and new vertex position $newVertex_i$ for each vertex and each facial feature in the 3D model. We set parameter p in linear interpolation as vertex weight defined for each facial feature. We calculate new vertex coordinate from old coordinate by scaling and translating according to facial feature transformations. Final coordinates of each vertex are obtained by applying equation 3 to it for each facial feature transformation (scale, translate), where vertex weight for current vertex and facial feature is set as parameter p. The vertex position $newVertex_i$ is calculated by concrete facial feature transformation.

After performing all local transformations according to facial features positions and sizes in images we have adjusted 3D head geometry, which shapes a head from input images. Then we create a head texture from detected head image areas and map this texture on the reconstructed geometry. Front and profile images are joined together in texture creation phase. The first step is scaling a smaller image up to have same head height as in the bigger one. Then is front head image rectangle from frontal image joined with profile head image rectangle. A profile rectangle is connected to front rectangle from both left and right sides. Horizontally flipped profile image is joined from the left side. X joint coordinate is in front image left eye left border position and right eye right position and eye center in profile image. Blending between images is calculated by the linear interpolation in connection area. After image junction, known texture part is cloned into background texture part. For example the hair texture is cloned into the space above head.

When the texture creation is finished, we need to calculate normal map texture. Grayscale copy of color texture is used as a surface height map, because it contains main head surface details information. For each pixel in grayscale image it is calculated its x direction and y direction intensity difference with the next pixel in current direction. This difference is scaled with surface height scale constant to decrease differences. We found empirically the height scale constant 0.01 in our experiments. X and Y differences are brought into account as derivations of image function in their directions. From these derivations we can create direction vectors and calculate image function normal as their cross product. Then we normalize obtained normal and transform its coordinates into RGB color space for storing in normal map texture.

The head model uv coordinates also need to be adjusted to correctly shape head in new created texture. For this purpose uv coordinates of vertices are recalculated for each facial feature. Scale and translation are performed on all vertices with the origin in concrete facial feature center and with linear falloff.

3.4 Head Rendering

When the 3D head model is reconstructed we would like to display it to user and enable him or her to interact with the model and optionally change parameters to reach better precision. For making the reconstructed 3D head model rendering more realistic we can use local lighting illumination calculation. Very simple approximation of skin illumination is a local Phong lighting model [24], because it is not so computational expensive and gives us a better performance with slower computers. To enhance details in rendering we can use a Normal mapping [13] technique, which simulates local object normals with a normal map texture calculated in the reconstruction phase. The Normal mapping enhances details, but causes very hard and rough surface, because it does not calculate light transport under the skin surface.

When we want to reach a realistic skin lighting, we should simulate a subsurface scattering effect caused by light scattering under the skin. We use a texture-space diffusion technique described by Simon Green in the GPU Gems book [11]. In the first step the 3D head model is rendered without the texture to texture space according to uv coordinates and lighting is calculated by a Phong model or advanced lighting calculations like the normal mapping. In the second step the result is blurred with a predefined convolution to simulate light transport under the skin. The Uniform blur only approximates subsurface light transport, because a distance between two pixels is in the texture space different than in the 3D world space. In the third step the model is rendered with the color texture and lighting computed in previous steps. The Light transport of different color channels could be simulated variously to create inner material light coloring, like soft red color from blood under the skin. Rendering techniques

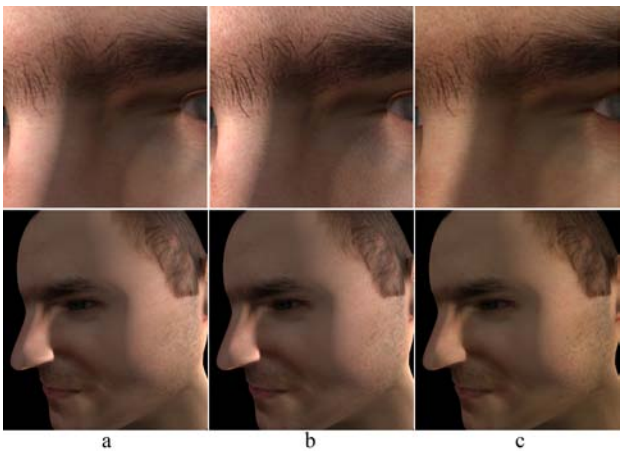


Figure 10: Head rendering result with techniques (a) Phong illumination model, (b) normal mapping, (c) texture-space diffusion. Detail rendering (top) and whole head (bottom).

results comparison could be seen in Figure 10.

4 Results

The proposed modeling system was implemented in C++ programming language using OpenCV [2] library in detection phase and OpenGL [3] library to render the reconstructed head model. Glsl shading language [4] was used to implement shaders calculating Phong model, normal mapping and texture-space diffusion lighting simulation.

Lighting calculation	fps
No shader	65
Phong lighting model	64
Normal mapping	42
Texture-space diffusion	26

Table 1: Frame rates in rendering the reconstructed 3D head model with different rendering techniques.

Method	Quality	Reconstruction time
Morphable Model	89%	50 minutes
CyberExtruder	82%	4 seconds
Our method	57%	4 seconds

Table 2: Reconstruction quality evaluation and speed comparison. Our solution, morphable model [8] and CyberExtruder [16] solution were tested.

The system was tested on AMD Athlon X2 1.9 Ghz, 2GB RAM computer with NVIDIA GeForce 7150 graphic card. Detection in a front image takes 1.54 seconds in average and detection in a profile image takes 1.8 seconds in average in this hardware configuration. The 3D head model updating process takes 4 seconds on average. The Reconstructed 3D head model was rendered at interactive framerates reported in Table 1. In each tested rendering technique we used the textured model.

Quality of reconstruction was measured by statistics of subjective human evaluation. Respondents were asked to evaluate the quality of 3D head reconstruction from concrete front and profile images in percentage. Three different approaches were compared by this evaluation (Table 2). Final approach quality was figured out as an average of respondents answers. Quality of our solution could be improved in future work by adding more head attributes to parameterized model and by enhancing detection precision.

Example of front and profile source photographs, reconstructed head geometry and textured model with Phong illumination could be seen in Figure 11.

We create a 3D head model from 2D images to use reconstruction result in various applications and therefore we need to export model and texture to common data format usable in broad scale of programs. Many data formats to store 3D information are available today, but most of them are strongly application dependent and their binary formats are difficult to read and do not allow all required information storing.

Therefore, for 3D data storage we chose Collada format [7, 5], which was created to enable work with one content in many digital content creation tools, it was accepted as industry standard by Khronos Group and it is used in OpenGL ES and several other real-time APIs. Collada provides us easy readable XML based data format to store required geometric and radiometric information.

The basic model information like reconstructed geometry, global transformations and visual effects should be stored in the output file. Collada output file contains relative links defining file names and paths to textures stored in jpg file format.

5 Conclusion and Future Work

We created the automatic image based 3D head modeling system, which is capable of reconstruct the 3D head model from 2D images. This system creates 3D head model automatically, but it allows user to change reconstructed model to reach better precision. The 3D head modeling system renders the reconstruction result with advanced illumination techniques simulating skin illumination. Finally, user can export reconstructed high polygonal model for use in other applications.

New parameterized head model based on the hierarchical tree of facial features, which defines vertex weights for each feature to set transformation intensity, was developed in this work. This model provides easy feature position and attributes estimation for modeling system and user. Our parameterized model consists of 32672 triangles and allows modeling system to reconstruct high resolution

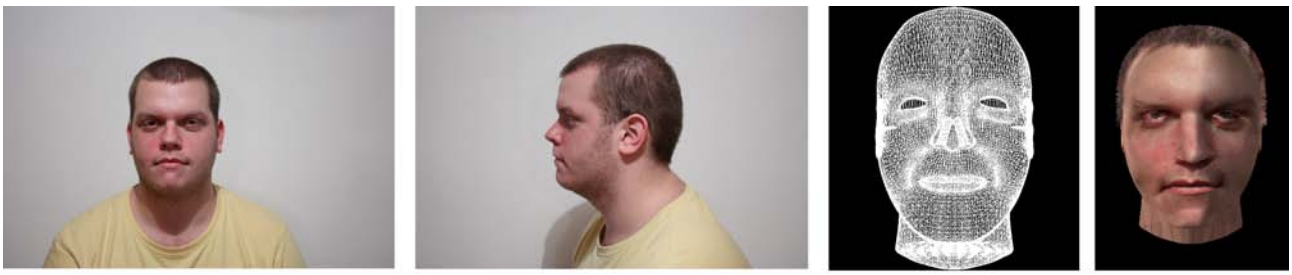


Figure 11: Example of front (a) and profile (b) source photograph, reconstructed head geometry (c) and textured model(d).

model.

We introduce a novel method for detecting head parameters from image with use Haar Cascade Classifiers and skin-tone based image segmentation in this work. This method gives us correct results to achieve good precision with updating a parameterized model.

To load parameterized head geometry and render the reconstructed model we created a Collada rendering engine, which could be used in many other works to easily load and render digital content with GPU.

Many extensions could further improve our modeling system, because the application is object based and allows updating or changing each reconstruction part for other purposes. More facial features controllers could be added and better head parameters detectors could be implemented to enhance the reconstruction precision and speed.

In a future work we are planning to improve the reconstruction precision with optical flow based techniques used in the morphable model approach [8]. Reconstruction precision could be improved also by adding more head attributes to the parameterized model. We are aiming to develop a hair reconstruction technique, which could update the modeling system to give full head and hair reconstruction result. Lower precision head model could be created by decimating high precision reconstructed model, but a better way is to estimate it during reconstruction and make model details selection part of the modeling system. We are planning to use OpenCL [1] library to move computation to GPU.

6 Acknowledgments

I would like thank doc. RNDr. Andrej Ferko, PhD. for his helpful guidance and support in this work, Jörgen Ahlberg for providing his parameterized face model report and Michal Šukola, Tomáš Matúš and others for providing their photographs to development and presentation purposes. I thank to anonymous reviewers for their many valuable comments and suggestions. This work has in part been funded by Slovak Ministry of Education VEGA No. 1/0763/09.

References

[1] Opencl. [online]. [quot. 2010-02-16]. Available on www: < http://www.khronos.org/opencl/>.

[2] Opencv. [online]. [quot. 2009-11-18]. Available on www: <http://opencv.willowgarage.com/wiki/Welcome>.

[3] Opengl. [online]. [quot. 2009-11-16]. Available on www: <http://www.opengl.org/>.

[4] Opengl shading language. [online]. [quot. 2009-11-17]. Available on www: <http://www.opengl.org/documentation/glsl/>.

[5] Collada - digital asset exchange schema for interactive 3d. [online], 2009. [quot. 2009-12-10]. Available on www: <http://www.khronos.org/collada/>.

[6] J. Ahlberg. Candide-3 an updated parameterized face. report no. lith-isy-r-2326. Technical report, Dept. of Electrical Engineering, Linköping University, Sweden, 2001.

[7] R. Arnaud and M.C. Barnes. *COLLADA Sailing the Gulf of 3D Digital Content Creation*. A K Peters, Wellesley, 2006.

[8] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[9] M. Castrillón-Santana, O. Déniz-Suárez, L. Antón-Canalís, and J. Lorenzo-Navarro. Face and facial feature detection evaluation, performance evaluation of public domain haar detectors for face and facial feature detection. In *International Conference on Computer Vision Theory and Application*, 2008. [online], [quot 2010-01-10]. Available on www: <http://alereimondo.no-ip.org/OpenCV/uploads/37/CameraReadyPaper63.pdf>.

[10] N. Faggian, A. Paplinski, and J. Sherrah. Active appearance models for automatic fitting of 3d morphable models. In *AVSS '06: Proceedings of the IEEE International Conference on Video and Signal Based Surveillance*, page 90, Washington, DC, USA, 2006. IEEE Computer Society.

[11] S. Green. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter Real-Time Approximations to Subsurface Scattering. Addison-Wesley Professional, Indiana, 2004.

[12] A. Hadid, M. Pietikäinene, and B. Martinkauppi. Color-based face detection using skin locus model and hierarchical filtering. In *ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 4*, pages 196–200, Washington, DC, USA, 2002. IEEE Computer Society.

[13] W. Heidrich and H.P. Seidel. Realistic, hardware-accelerated shading and lighting. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 171–178, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[14] R.L. Hsu, M. Abdel-Mottaleb, and A.K. Jain. Face detection in color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:696–706, 2002.

[15] Y. Hu, B. Yin, S. Cheng, and Ch. Gu. An improved morphable model for 3d face synthesis. In *Proceedings of International Conference on Machine Learning and Cybernetics*, pages 4362–4367, Washington, 2004. IEEE Computer Society.

[16] J.D. Ives. Method for generating a three-dimensional representation from a single two-dimensional image and its practical uses. [online]. Newark: CyberExtruder.com, Inc. [quot. 2009-12-11]. Available on www: <http://www.cyberextruder.com/>.

[17] Ch. Kim and J. Yi. An optimal chrominance plane in the rgb color space for skin color segmentation. In *International Journal of Information Technology*, volume 12, pages 73–81. World Scientific Publishing, 2006.

[18] P. Kuchi, P. Gabbur, P.S. Bhat, S.David, and S. Smieee. Human face detection and tracking using skin color modeling and connected component operators. In *IETE journal of research*, volume 48, pages 289–293, New Delhi, 2002. Institution of Electronics and Telecommunication Engineers.

[19] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002*, pages 900–903, 2002.

[20] K. Lin, F. Wang, J. Yao, and Ch. Zhou. Human head modeling based on an improved generic model. In *FSKD '08: Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, pages 300–304, Washington, DC, USA, 2008. IEEE Computer Society.

[21] Z. Liu, Z. Zhang, C. Jacobs, and M. Cohen. Rapid modeling of animated faces from video. Technical report, Journal of Visualization and Compute Animation, 2000.

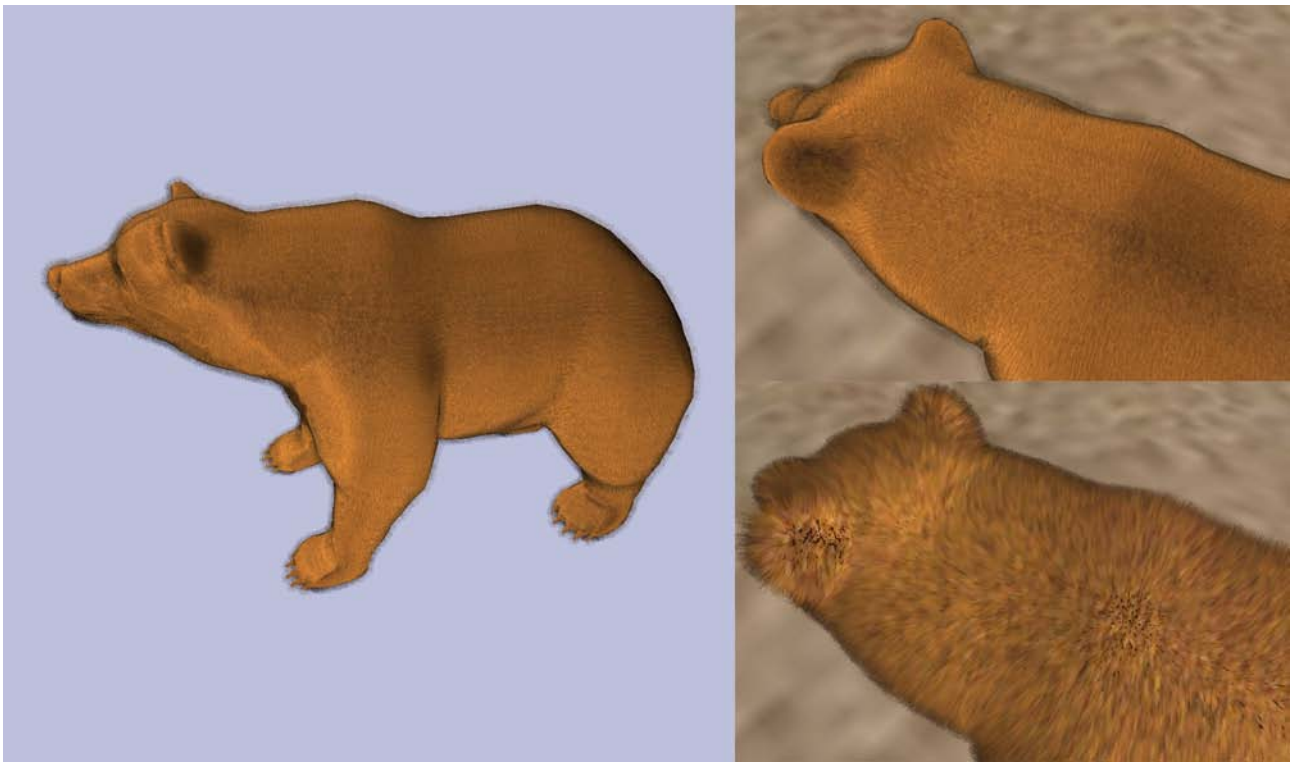
[22] M. Rydfalk. Candide, a parameterized face, report no. lith-isy-i-866. Technical report, Dept. of Electrical Engineering, Linköping University, Sweden, 1987.

[23] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Conference On Computer Vision And Pattern Recognition*, pages 511–518, 2001.

[24] J. Žára, B. Beneš, J. Sochor, and P. Felkel. *Moderní počítačová grafika*. Computer Press, Brno, 2004.

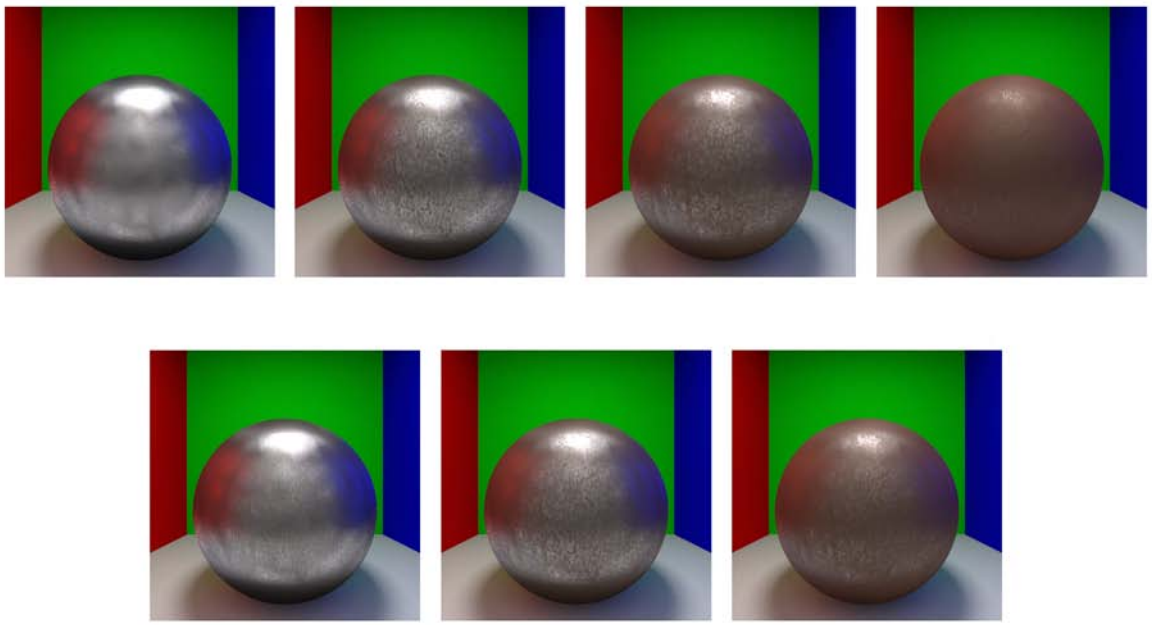
Color Plates

Martin Berger
Real-time Fur Using GPU-based Raycasting
pp. 11–17



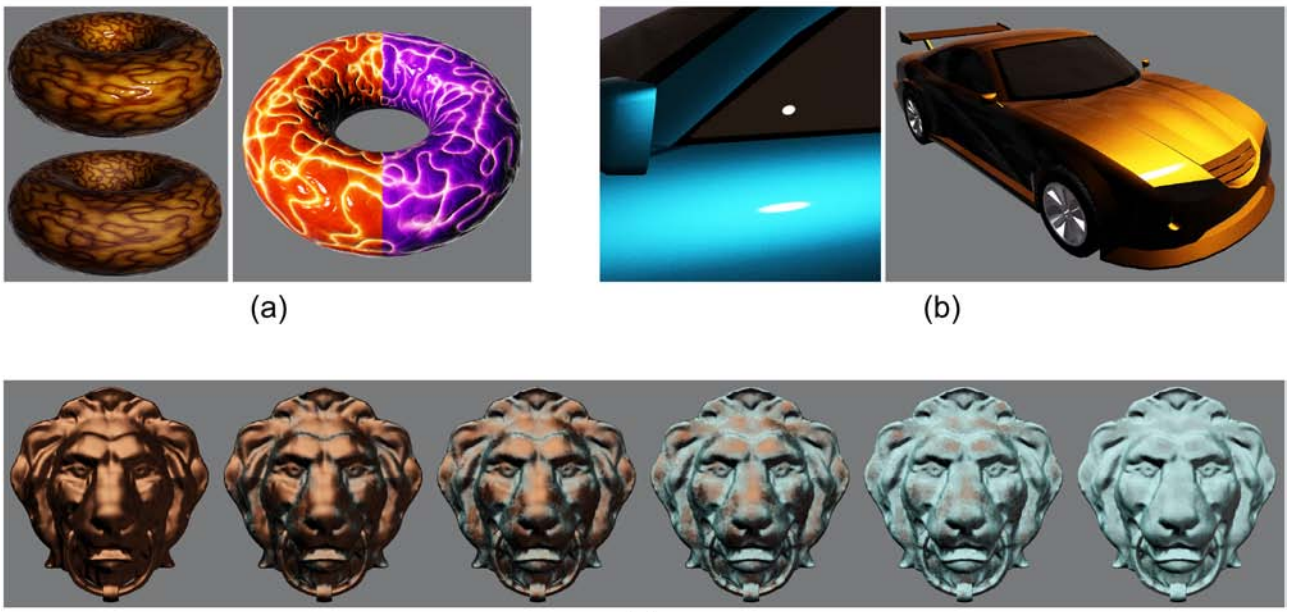
Left: A model of a bear rendered with our technique using 128 planes per tangent space axis. Right: A close-up comparison of our technique with textured shells.

T. Langenbucher, S. Merzbach, D. Möller, S. Ochmann, R. Vock, W. Warnecke, and M. Zschippig
Time-Varying BTFs
pp. 19–25



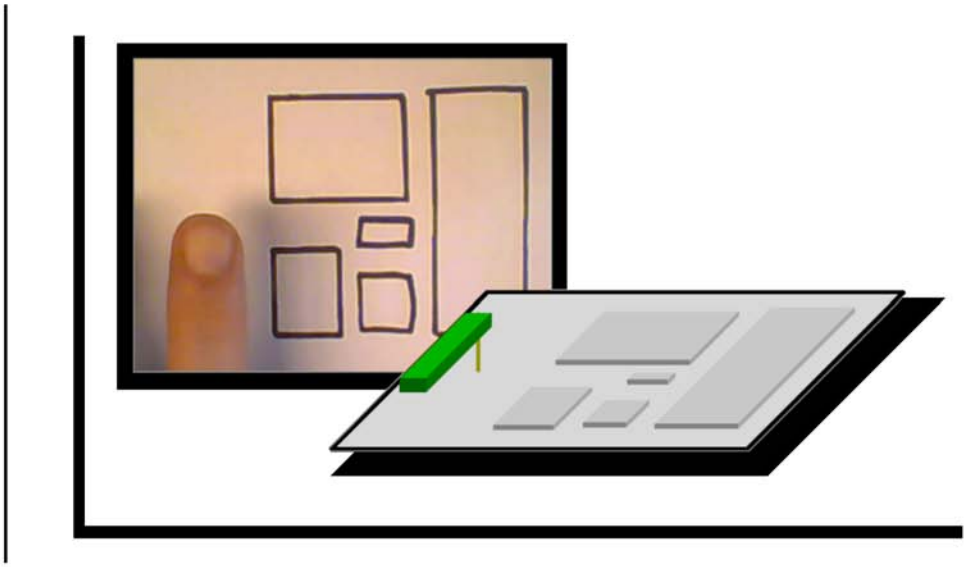
Renderings of a Time-Varying Bidirectional Texture Function (TV-BTF) of rusting metal. Top row: PCA-compressed BTFs of the original four aging stages of the material. Bottom row: linear interpolations between the compressed BTFs, demonstrating the concept of TV-BTFs.

Oskar Elek
Layered Materials in Real-Time Rendering
pp. 27–34



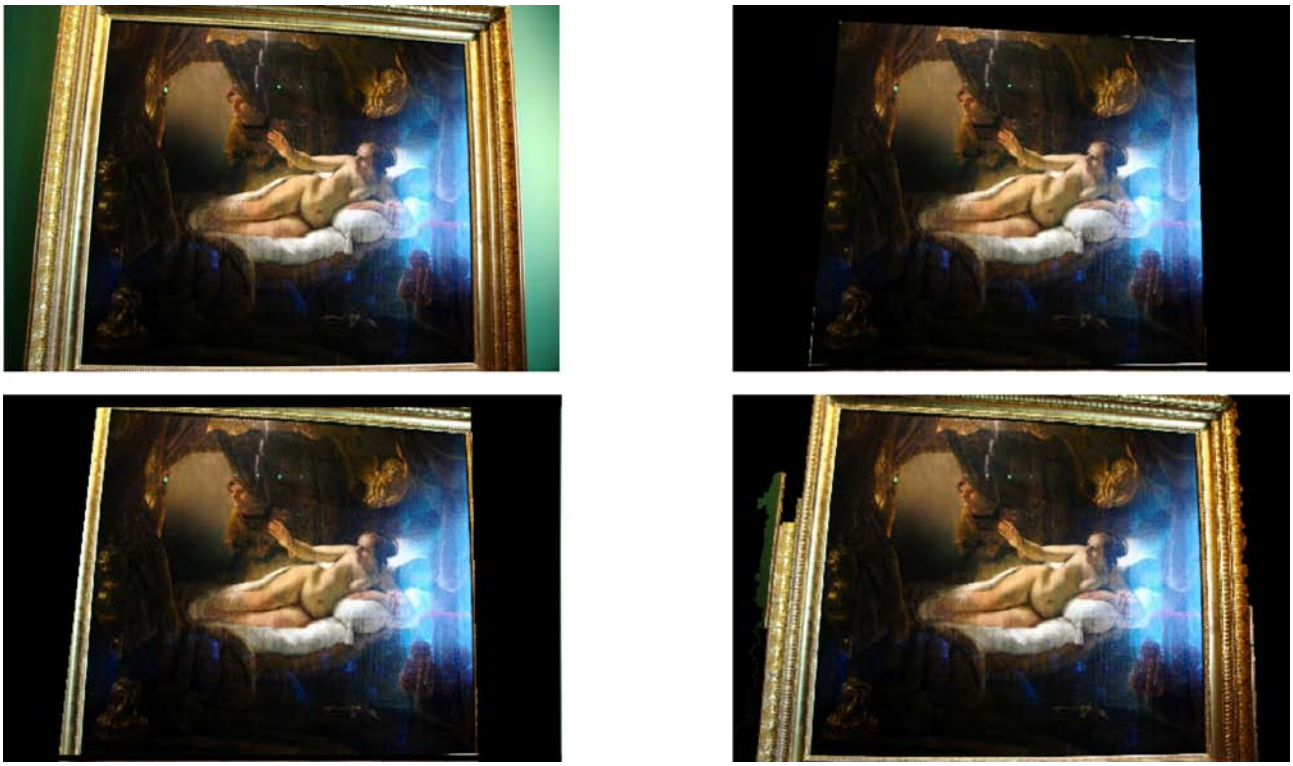
(a) Examples of the general layered model - varying top layer roughness and thickness. (b) Specialized metallic car paint model with sparkling effect. (c) Temporal development of patina on a copper object.

Pavel Vlašánek
Usage of the Webcam as 3D Input Device
pp. 45–50



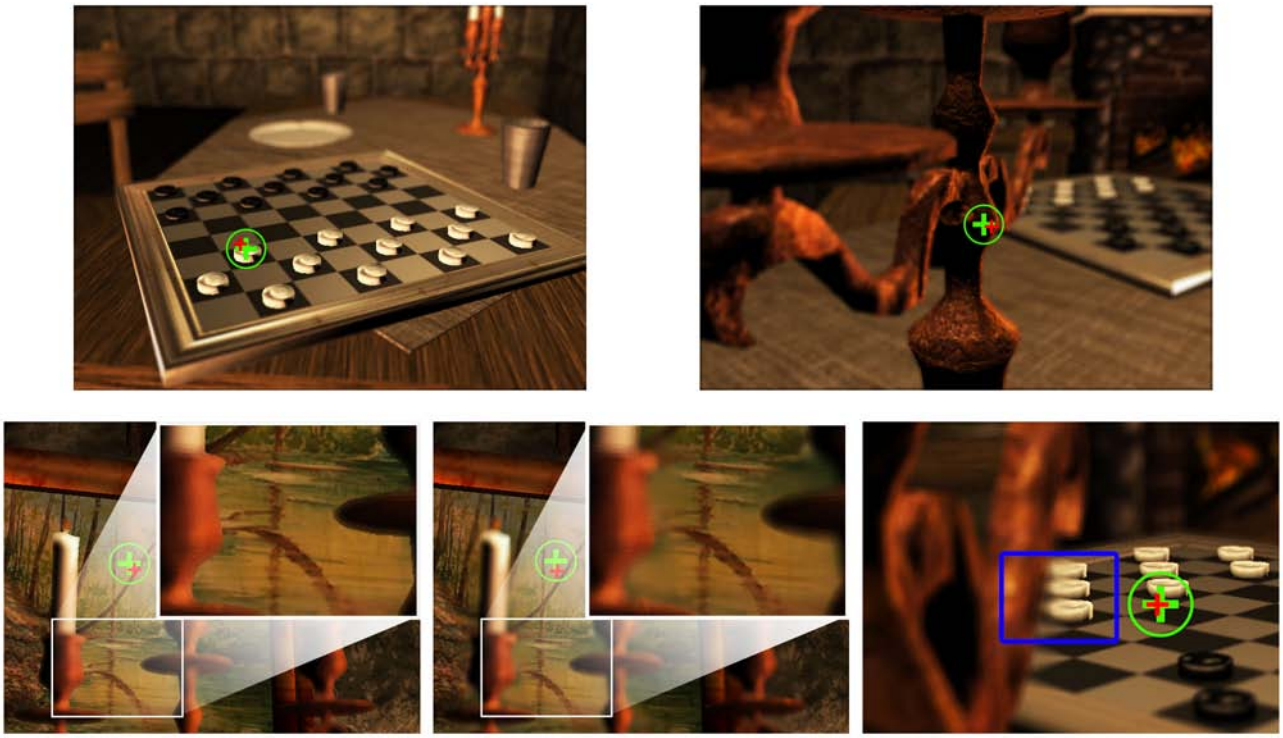
The system is able to recognize a hand drawn keyboard and compute a position of the finger in the space.

Zuzana Haladová
Segmentation and Classification of Fine Art Paintings
pp. 59–65



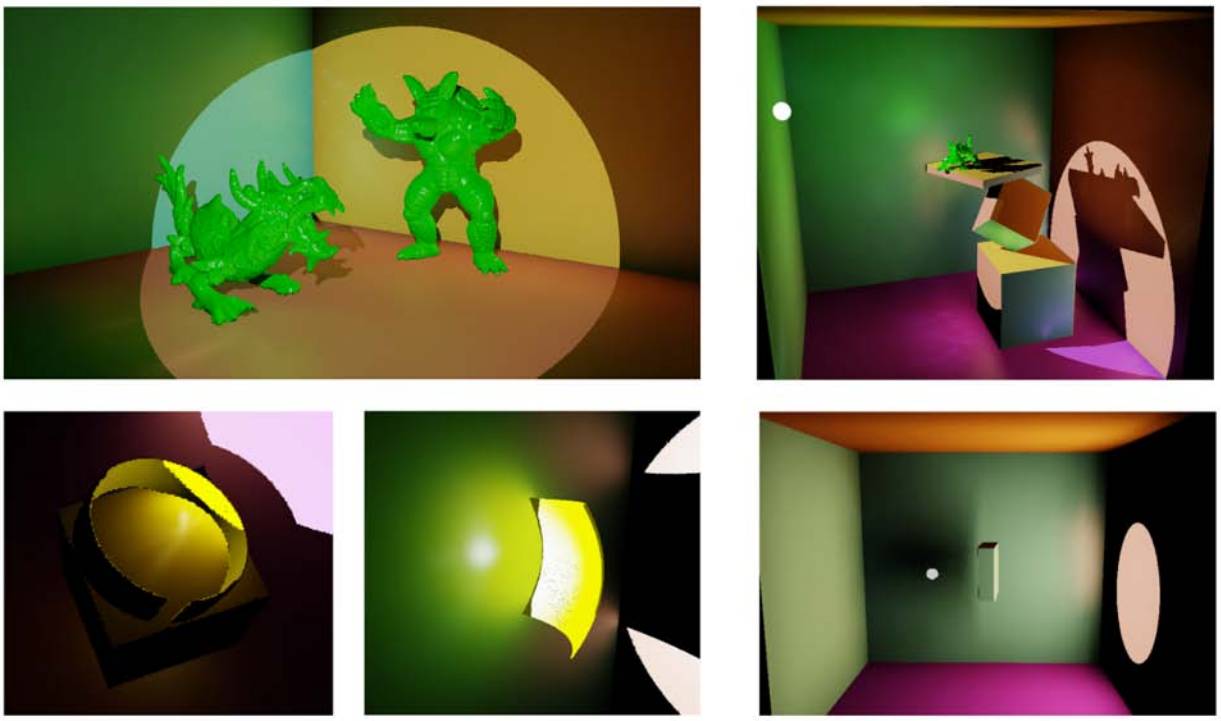
Original image and images of paintings segmented with Anisotropic diffusion, Gauss gradient and watershed methods.

Bartosz Bazyluk
Eye Tracking in Virtual Environments: Implementation of Gaze-point Dependent Depth of Field
pp. 75–81



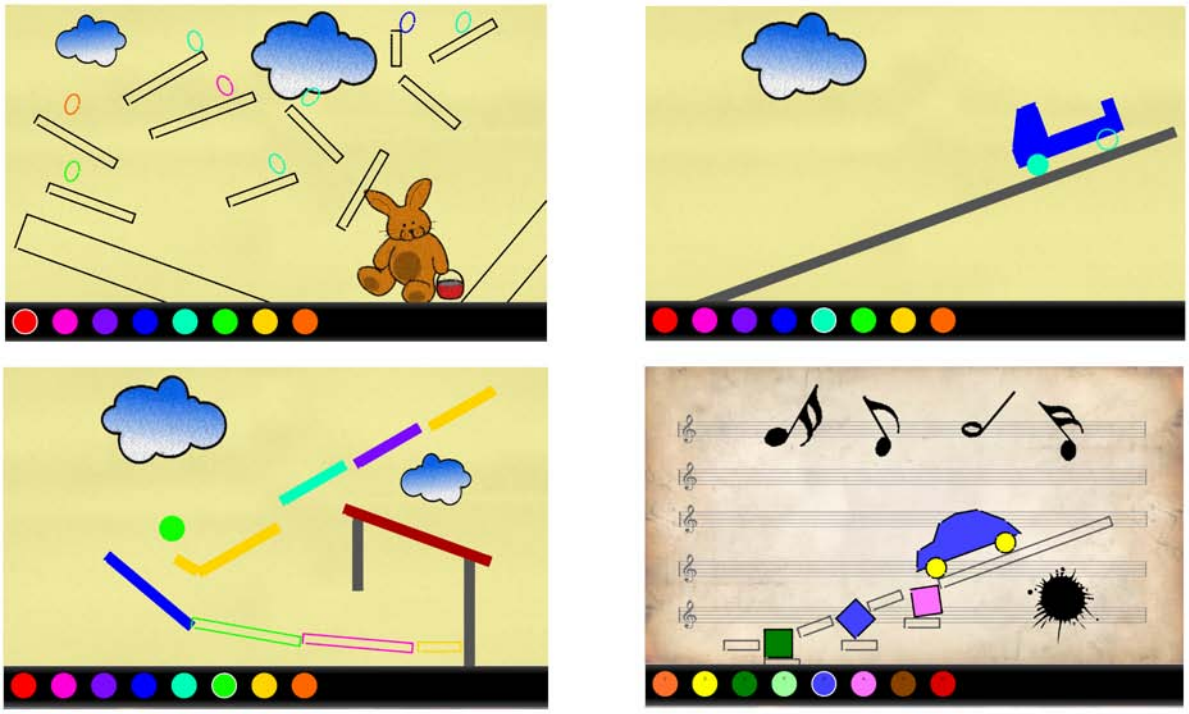
Top row: Example screens from our demo that present the resulting depth of field effect. Bottom row: The depth discontinuity problem (left) and the solultion (middle) with its downside (right). On all figures the green crosshair represents the filtered, while red cross is the raw gaze point.

Reinhold Preiner
Real-Time Global Illumination in Point Clouds
pp. 83–90



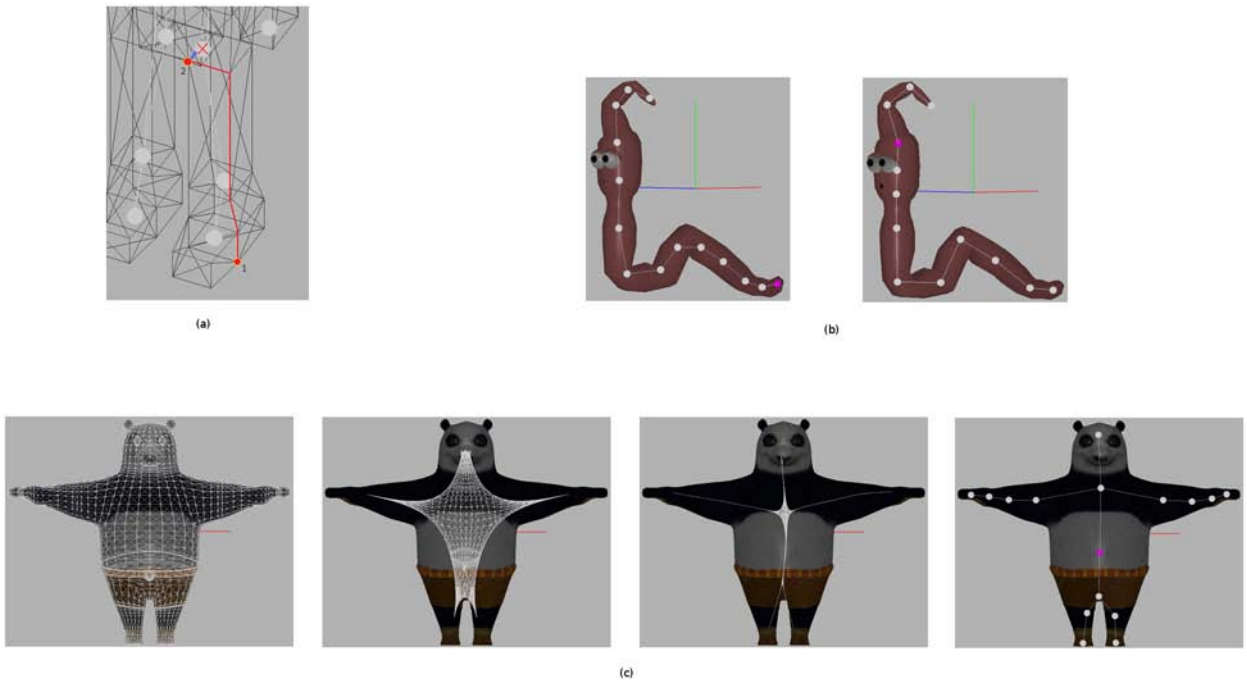
Left-Top: Two figures illuminated by our GI technique. Right-Top: Cornell-Box scene showing specular reflections. Left-Bottom: Caustics produces by our technique. Right-Bottom: Sample scene showing an indirect shadow.

Tomáš Pastorek
Concept of Interactive Coloring book
pp. 101–107



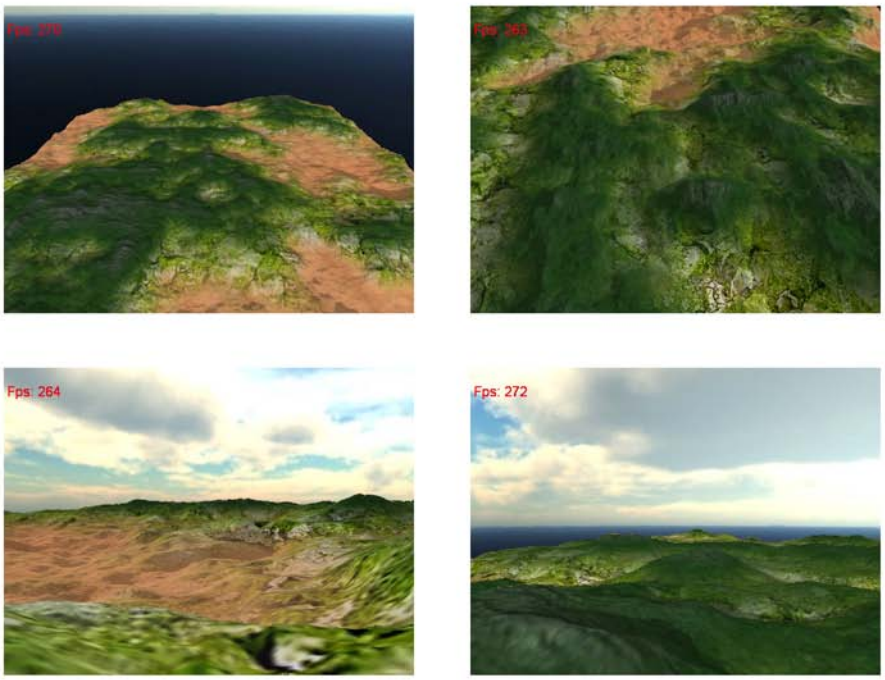
Scenes from Interactive Coloringbook, a programmable physics based game environment for small children. Left-Top: Easter eggs scene; Right-Top: Truck scene; Left-Bottom: Path Coloring; Right-Bottom: Music scene.

Martin Madaras
Extraction of Skinning Data by Mesh Contraction with Collada 1.5 Support
pp. 133–139



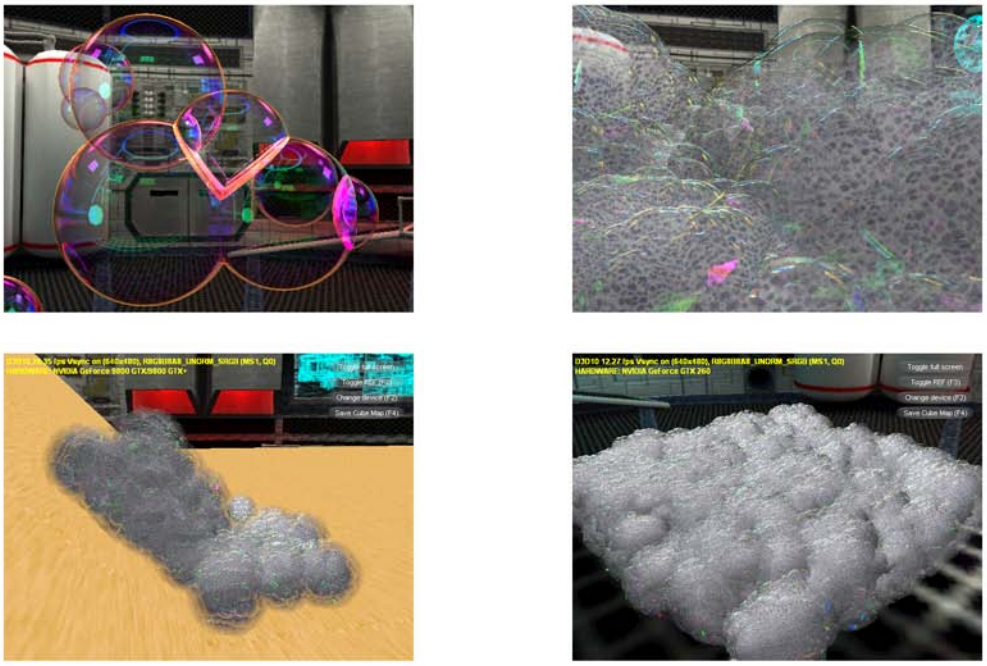
(a) Computation of the geodesic distance between the control point marked with red cross and the mesh vertex. (b) Comparison of the extracted skeleton (left) and the skeleton rigged by an artist (right). (c) Example of the contraction of the geometry and the extracted skeleton.

Zsolt Fehér
Terrain Rendering with the Combination of Mesh Simplification and Displacement Mapping
pp. 141–147



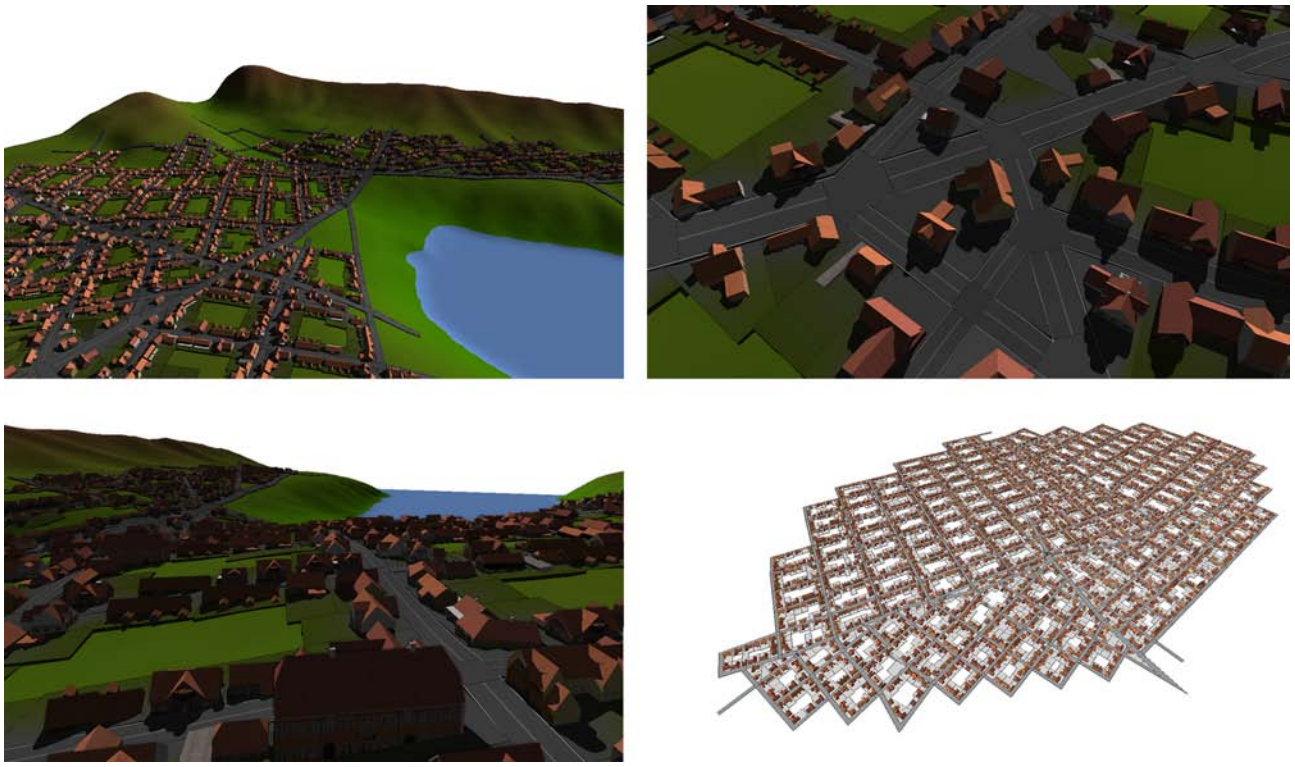
These four viewpoints were selected to be used during frame rate test of the algorithm on different platforms with different options. Shown frame rates were achieved on Geforce 8800 GT, 2049×2049 height map, 256×256 quads.

Tamás Huszár
GPU-Supported Bubble and Foam Rendering
pp. 149–155



Top left: bubble formation with mutual bubble walls. Top right: close-up of rendered foam Bottom left: foam formation sliding down on a slope. Bottom right: large foam consisting of more than 180000 bubbles.

Johannes Scharl
A Constraint Based System to Populate Procedurally Modeled Cities with Buildings
pp. 157–164



Some results from our system. Top left: a small village on a hillside next to a bay. Top right: Close-up of a more complex junction in the village. Bottom left: Close-up with view over the bay to the hillside where streets spread over the plain of the terrace. Bottom right: a larger city without terrain with far over 100 streets and more than 2.500 buildings.

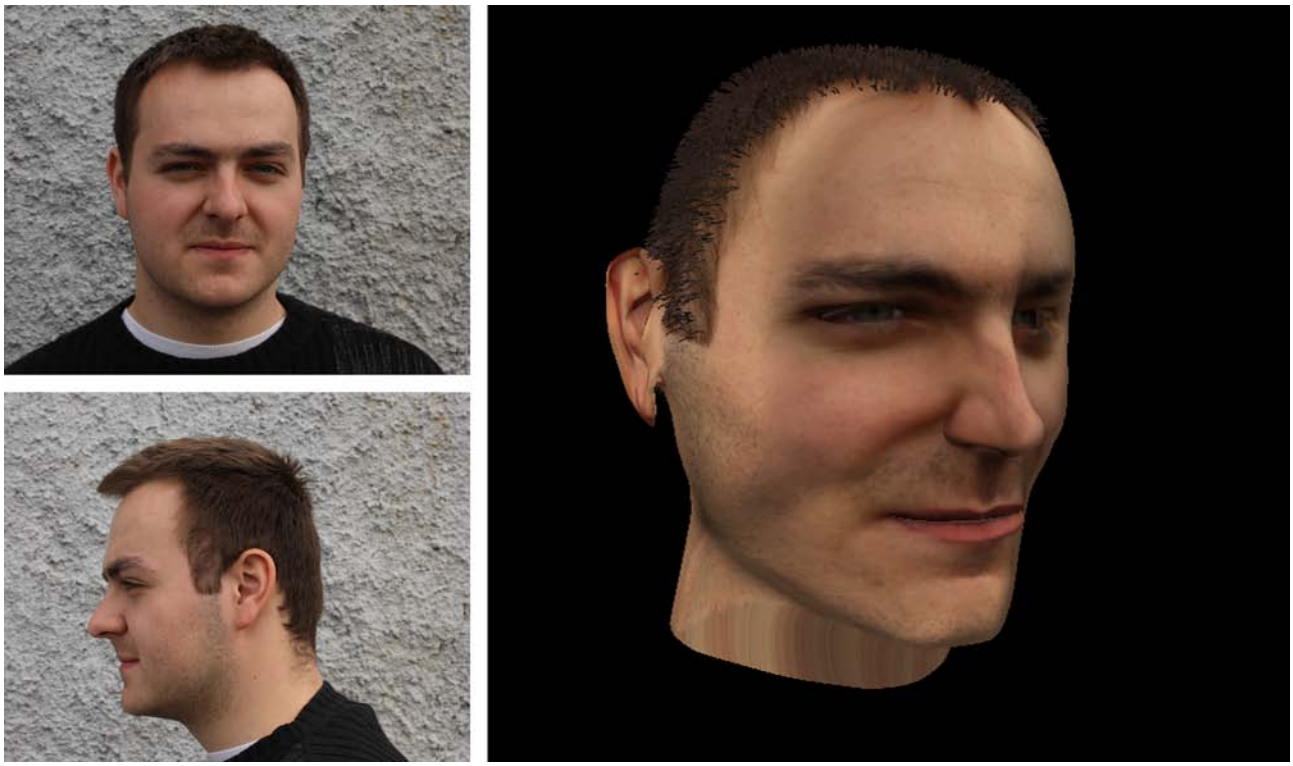
Goran Radosevic
Laser Scanning Versus Photogrammetry Combined with Manual Post-modeling in Stecak Digitization
pp. 167–174

Left: The final model of stecak created using photogrammetry technique only. Right: The final model of stecak improved in 3ds Max. This model has maximum tested level of details.

Bartłomiej Specjalny
The Prototype Light Projection System for Cultural Heritage Reconstruction
pp. 183–190

Left: A photograph of a sample plate of the Wkryujście Altar titled: Jesus Christ coronation with thorn. Middle: The test object used to prototype the reconstruction technique. Right: The final mesh reconstructed from the point cloud.

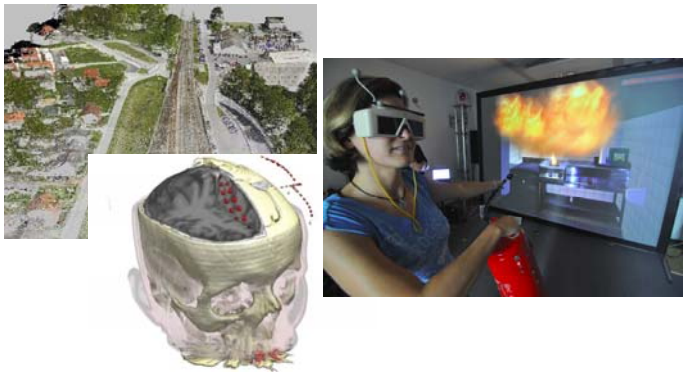
Peter Kán
Automatic Image-Based 3D Head Modeling with Parameterized Model Based on Hierarchical Tree of Facial Features
pp. 191–198



Example of source photographs (Left) and the reconstructed 3D head model (Right) rendered with the texture-space diffusion technique.

Advertisments for Sponsors of CESCG 2010

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

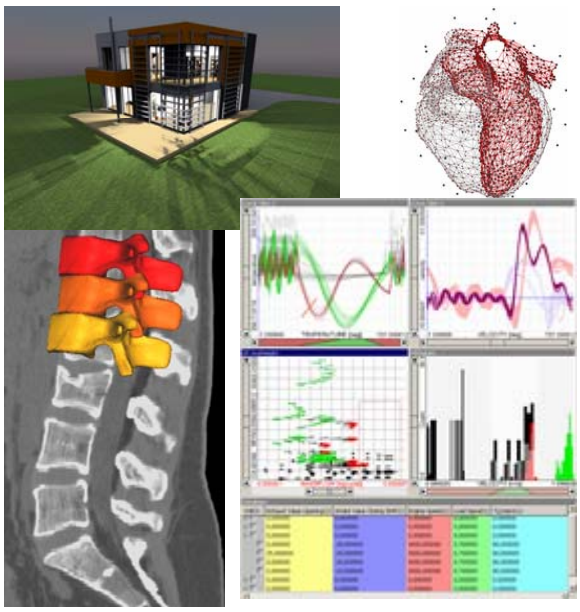
Scientific Partners of VRVis:

- 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.

sféra®

Grafické informačné systémy

spokojnosť zákazníkov
+
kvalita produktov
=
priestor pre váš úspech

Ponuka služieb a riešení:

- Poradenstvo a analýzy v energetike
- Obchodovanie a hospodárenie s energiami
- Správa obchodných meraní
- Správa technologických rozvodov
- Správa majetku a riadenie údržby
- Správa životného cyklu zariadení
- Správa technickej dokumentácie
- Správa a údržba cestnej siete
- Projektovanie energetických rozvodov
- Mapové diela
- Akreditované certifikačné služby
- Spracovanie technických, grafických a CAD dát
- Predaj a inštalácia softvéru a hardvéru

www.sfera.sk

ISO 9001:2000
DNV

CSA SYSTEMS

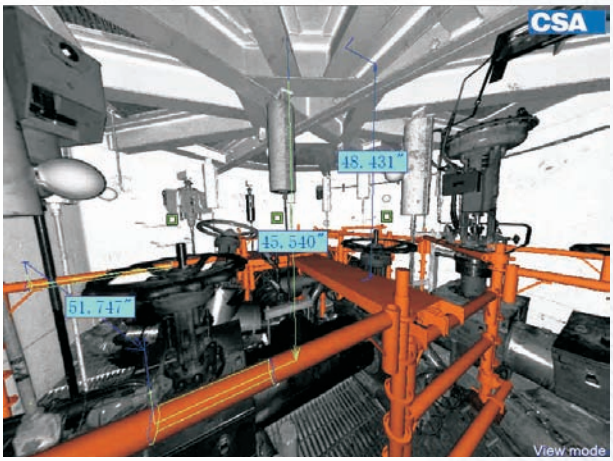
American-slovak software company established in 1992

The core of our activities includes:

development and implementation of lifecycle facility management systems

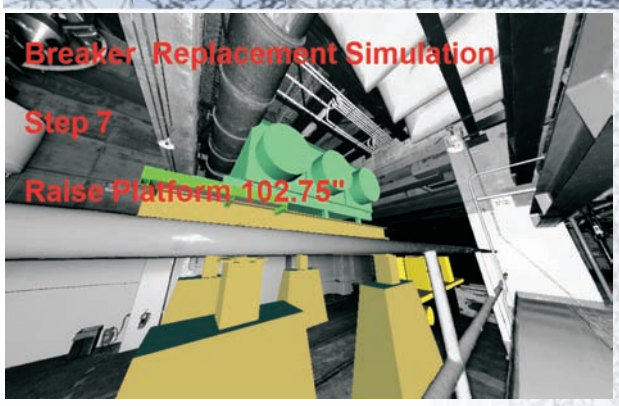
3D laser scanning technology

CAD databases conversions



Integrate 3D scaffolding model with scan data.

Radiological data displayed on PanoMap scans in real time.



Simulate equipment removal / replacement with 3D animation. Verify for clash detection.

CSA Systems s.r.o.
Nábrežná 4
971 01 Prievidza

Tel.: +421 46 5430572
Fax: +421 46 5430834
E-mail:csa@csasystems.sk
Web:www.csasystems.sk

Túto konferenciu sme podporili z lásky k ~~vede~~ vedcom.

www.eea.sk