Proceedings of the

15th Central European Seminar on Computer Graphics

May 2 - 4, 2011
Viničné, 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

Edited by Michael Wimmer, Jiří Hladůvka, and Martin Ilčík © 2011
ISBN: 978-3-9502533-3-7

Impressum

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

ISBN 978-3-9502533-3-7

Welcome to CESCG 2011!

This book contains the proceedings of the 15th Central European Seminar on Computer Graphics, short CESCG, which continues a history of very successful seminars. Again this year, CESCG proceedings have an ISBN (978-3-9502533-3-7) and will therefore remain retrievable as long as there are libraries!

The long history of CESCG has started in 1997 in a medium-sized lecture room in Bratislava, bringing together students from Bratislava, Brno, Budapest, Graz, Prague, and Vienna. The idea found wide appraisal and the seminar moved to the beautiful castle of Budmerice, where it was held for 8 consecutive years, constantly growing in size and attraction. It was just in the 10th anniversary year 2006 that CESCG had to take a detour to move to Častá-Papiernička Centre, while it was back in Budmerice castle since 2007. Unfortunately, this year another anniversary has to be celebrated outside of the castle in an old mill in Viničné.

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 **Martin Ilčík** for taking care of the complete reviewing process and scientific program preparation. We are very thankful to the CESCG organizers from Bratislava, mainly **Andrej Ferko**, always an inspiration to CESCG; and Matej Novotný, Ela Šikudová, Iwka Varhaníková (for Wienislava exhibition), David Běhal, Janka Dadová, Martin Florek, Stanislav Stanek, Roman Ďurikovič, and Ján Lacko for the excellent preparations and on-site organization.

The main idea of CESCG is to bring students of computer graphics together across boundaries of universities and countries. Therefore we are proud to state that we have achieved again a very high number of 13 participating institutions and a very tight time schedule of 20 valuable student works and two invited talks. We welcome groups from Bratislava (UK and STU), Slovakia; Brno and Prague (CTU and KU), Czech Republic; Budapest, Hungary; Bonn, Germany; Graz and Vienna, Austria; Szczecin, Poland; Warwick, United Kingdom; Maribor, Slovenia; and Sarajevo (Univ. and SSST), Bosnia and Herzegovina.

We assembled a large International Program Committee of 16 members, allowing us to have each paper reviewed by three IPC members during the informal reviewing process. We would like to thank the members of the IPC for their contribution to the reviewing process. The IPC of CESCG 2011 consists of:

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

The first invited talk "Realistic Rendering of Natural Phenomena" will be held by Tomoyuki Nishita from Department of Complexity Science and Engineering of University of Tokyo, Japan. The second invited talk by Chris Wojtan from Computer Graphics Group of Institute of Science and Technology, Austria, will be about "Deformable Surfaces with Topology Changes for Physics-Based Animation".

The seminar is held under the auspices of the Austrian Ambassador to Slovakia, His Excellency **Dr. Josef Marcus Wuketich**, and is co-organized with the Spring Conference on Computer Graphics (SCCG), which takes place before 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 2011:

- OCG, the Austrian Computer Association
- VRVis, a research center for virtual reality and visualization in Vienna
- KAUST, King Abdullah University of Science and Technology
- Qualcomm, wireless telecommunications research and development company
- Brno University of Technology, Faculty of Information Technology, Department of Computer Graphics and Multimedia
- Czech Technical University in Prague, Faculty of Electrical Engineering, Computer Graphics Group
- Graz University of Technology, Institute for Computer Graphics and Vision
- Johannes Kepler University in Linz, Institute for Applied Geometry
- The Ministry of Education, of the Slovak Republic
- SISp, Slovak Society for Computer Science

Please note that the electronic version of these proceedings is also available at http://www.cescg.org/CESCG-2011/.

April 2011,

Michael Wimmer
Jiří Hladůvka
Martin Ilčík

Table of Contents

Invited Talks

Lighting

Rendering

Attention & Entertainment

Human Computer Interfaces

Natural Phenomena & GPU

Visualization

Color Plates

Sponsors of CESCG 2011

Invited Talks

Realistic Rendering of Natural Phenomena

Tomoyuki Nishita

University of Tokyo
Japan

Abstract

The simulation of various natural phenomena is one of the important research fields in computer graphics. In particular, aspects such as clouds, water, smoke, and sands are indispensable for creating realistic images of natural scenes. Therefore, a lot of researchers have been trying to develop methods for simulating and rendering these scenes. In my presentation I focus on water, granular materials, clouds, atmospheric effects. These phenomena have the common feature that they are consist of the effects of small particles. And the color greatly depends on the properties of light scattering due to particles. To create realistic images, physical based simulation and rendering are required. In particular, the color greatly depends on the properties of light scattering due to particles. I would like to introduce efficient methods for creating realistic images of such natural phenomena. Recently, the results of particle-based simulations have been often visualized using a large number of metaballs. I introduce a fast technique for rendering metaballs on the GPU.

Deformable Surfaces with Topology Changes for Physics-Based Animation

Chris Wojtan

Institute of Science and Technology
Austria

Abstract

Accurate computational representations of highly deformable surfaces are indispensable in the fields of computer animation, medical simulation, computer vision, digital modeling, and computational physics. After reviewing common representations of deformable surfaces, I will present some of my recent contributions to the field of computer graphics.

I will first present results from an algorithm that generates highly detailed continuum mechanics animations by combining a finite element method with a tetrahedral mesh generator and a high resolution surface mesh. Next, I will present an efficient solution for the challenging problem of computing topological changes between detailed surface meshes, allowing us to track surfaces in computational fluid dynamics applications with unprecedented levels of accuracy and detail. This surface tracking technique also opens the door for a unique coupling between surficial finite element methods and volumetric finite difference methods, in order to simulate liquid surface tension phenomena more efficiently than any previous method. Due to its dramatic increase in computational resolution and efficiency, this method yielded the first computer simulations of a fully developed crown splash with droplet pinch off.

Lighting

Augmented Reality platform for enhancing integration of virtual objects

Mohamed El-Zayat*
Supervised by: Selma Rizvic†

Faculty of Electrical Engineering, Sarajevo, Bosnia and Herzegovina

Abstract

With the wide spread of high end processors integrated in mobile devices, ranging from 1GHz processors, to dual core processors and hybrid processors (GPU and CPU on one chip), augmented reality became more popular solution for visualization and navigation. This paper proposes an augmented reality platform for organizing and enhancing integration of computer generated objects by introducing lights, shaders and shadows, in pursuing for better experience for the end user, emphasizing on outdoor environments.

Keywords: Augmented Reality, Platform, Enhancing Integration, Real Time sun Tracking

1 Introduction

Augmented reality (AR) is a relatively new and promising concept. The ability of superimposing digital elements on a physical world with means of interaction with the surrounding world is quite intriguing idea, since AR introduction in 1968 by Ivan Sutherland [4]. However, the technology by that time and for almost next 3 decades was quite limited to lab research, since the mobility nature of AR, and lack of capable mobile processors.

The rapid development of mobile GPUs, CPUs and recently hybrid processors, leads to an increase in popularity of the AR technology. Mobile devices play important role in AR technology as they combine processor, memory, display and interaction technology into one single device [14]. There are two main trends in AR research: registration, where researchers try to solve misalignment and world tracking problems; and integration, where researchers are directed towards the enhancement of computer generated object integration with the surrounding environment. This work proposes an AR mobile platform for enhancing integration of virtual objects in outdoor environments.

The rest of the paper is organized as follows: Section 2 gives an overview of the related work in the field; Section 3 illustrates the proposed AR mobile platform; Section 4 covers registration and Section 5 covers integration, proposing a real time sun tracking system for capturing the current lighting conditions of the environment. In Section 6 we present the algorithm. Section 7 presents results compared to other AR platforms. Finally, in Section 8 we conclude the paper and give some directions for future work.

2 Related Work

In recent years we have seen significant advances in two fields of user interface research: virtual environments, in which 3D displays and interaction devices immerse the user in a synthesized world, and mobile computing. Previous research in mobile AR has addressed a variety of application areas including 3D mobile AR systems for exploring urban environments [6], enhancing registration through making a hybrid registration for outdoor AR [1], improving teaching with mobile AR for learning and training [11], location based AR for indoor environments [9], enhanced computer generated objects rendering using environment illumination [7]. In pursuing better registration of AR objects, researchers are trying to combine computer vision with sensors for achieving more accurate results [5]. Additionally, a combined solution for illumination techniques for AR objects is discussed in [8]. One commercial platform that caught many mobile device users' attention is Layar AR browser [13] (Figure 1).

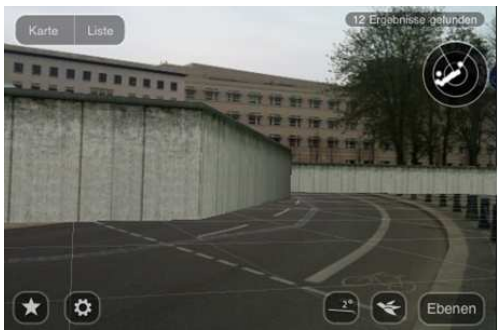


Figure 1: Layar Browser,the original appearance of the Berlin wall (image courtesy of layar.com)

This paper proposes the AR mobile platform for arranging AR objects with an emphasis on enhancing the integra-

*mohamed@fit.ba
†srizvic@etf.unsa.ba

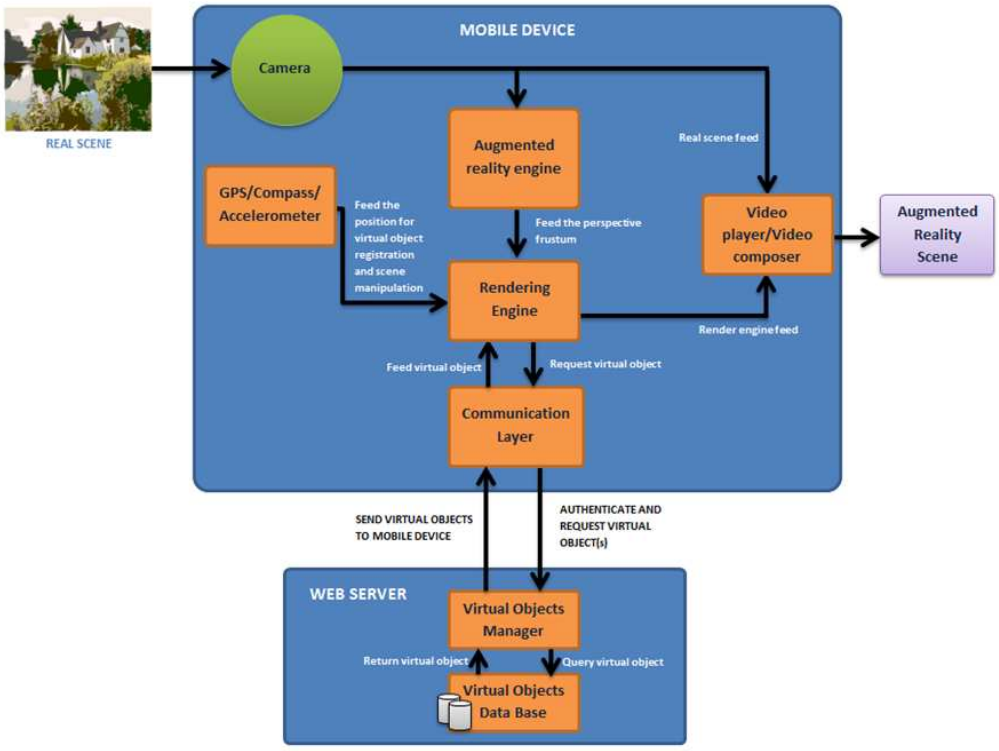


Figure 2: AR Mobile Platform architecture

tion of computer generated objects in the outdoor environment, by introducing lights and shaders to the augmented objects.

3 AR platform

Making AR systems that work outdoors is a natural step in the development of AR toward the ultimate goal of AR displays that can operate in any environment [2]. A user, walking outdoors, could see spatially located information directly displayed over live camera stream, helping to navigate and identify features of interest [1].

While designing the platform, the mobility and optimization factors were taken into consideration as described in the upcoming sections. Figure 2 shows the overall architecture of the platform.

In order to achieve higher performance and decrease the disk usage by 3D models and their associated textures, all 3D models are stored on a web server, and will be downloaded on the mobile device once the user is near the location where the virtual object should be displayed.

Comparing current revision of the proposed AR platform architecture (figure 2) and Layar architecture as shown in Figure 4, more support to developers and content providers is given from Layar, in contrast with the presented AR platform. Although the developers support was not considered in this current revision yet, one of the improvement ideas over Layar is saving 3D objects in relational SQL database as shown in Figure 3. This gives the opportunity to permitted organizations to run analysis and data mining techniques using off-the-shelf software to collect more data from users to further enhance the public service at certain most visited areas, thus further enhancing the proposed AR platform.

3.1 Server Side

A database containing 3D models along with the associated textures, and additional lighting data for improving the integration of the virtual object, resides on the server side. Lighting details are covered in Section 5.

The Virtual Objects Manager (VOM)is a web service responsible for handling authentication and requests to the database. This component would be crucial in case this platform is implemented for a mobile provider. In case that authentication is not needed, VOM will add the security layer needed to protect models database. 3D models are stored in a relational SQL database as shown in Figure 3:

3D models are indexed using their GPS coordinates. Geometry table contains the actual geometry data of the 3D model, stored as an array of doubles for each of Vertex, Vertex Normal and Texture Coordinate. Furthermore, the texture field holds compressed texture atlas for the geometry.

The illumination data (i.e. Artificial light) of the objects are stored in Lighting Data table, where color and position of the light are stored as array of doubles.

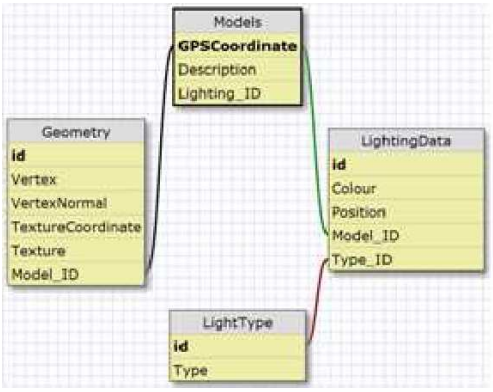


Figure 3: 3D models database

Organizing augmented objects in a relational SQL database provides the opportunity for other mobile devices/platforms that support connecting to SQL databases to benefit from the proposed AR platform. Therefore, it is possible to port the AR platform to wider variety of mobile devices.

Using the server side for managing virtual objects will relieve the users from constantly updating and downloading the complete application once 3D models get updated, thus increasing the performance and saving unnecessary storage load.

3.2 Client Side

This platform requires a mobile device that is GPS capable along with at least accelerometer and compass. Since most mobile device vendors are integrating these sensors as a standard in their devices, we believe that in few years these devices will be common among users.

Mobile devices contain two major components: registration and integration components. Registration component is responsible for AR registration, which could be done using a complete sensor based registration (i.e. combining GPS, accelerometer and compass data as discussed in Section 4), or hybrid one as discussed in [1]. A hybrid registration could combine some elements of computer vision and sensor data to improve the integration of computer generated objects. Therefore, the registration system could handle unstructured and unprepared environments [5], and in this case, the AR engine will be activated in the augmentation pipeline.

Once the user gets to a desired location detected by GPS, where a virtual object resides, registration component will generate the frustum that will be handed to the rendering engine. Rendering engine will send a request to the communication layer to load 3D object and its associated data from the server side, thus rendering the virtual object and ”clearing” the background with the camera feed, hence superimposing the virtual object over the physical world.

4 AR Registration

In order to enhance the integration of augmented objects, improving registration is required. GPS data is required to determine the position of the virtual object in the physical world and the position of the user according to the position of virtual object, hence calculating the position of the frustum according to the physical world using equation 1:

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 < r^2 \quad (1)$$

x_1, y_1 represents user position, while x_2, y_2 represents the virtual object's position. r is the range value .If the user is in the range of the detected object, he/she will be notified and the frustum will be generated.

In order to detect the rotation of the user, a compass will be used for azimuth rotation direction, and the accelerometer will control frustum altitude as shown in figure 5:

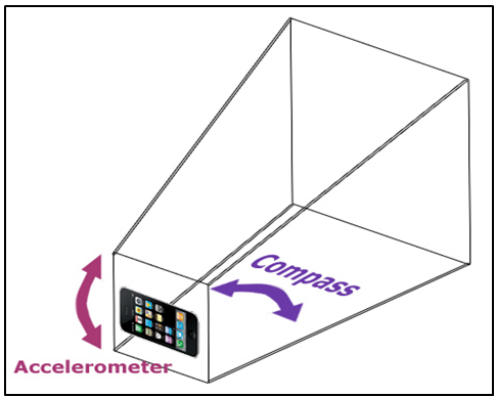


Figure 5: Frustum controlled by sensors

The registration is sensor based, similar to [14] and [13], in pursuing for saving CPU/GPU cycles to enhance the rendering of augmented objects, hence improving the integration. One of the registration types that are crucial for displaying correct imagery in augmented reality is precise alignment between the projected image and the features on the display surface [16]. Therefore, the elevation of the 3D object is determined from elevation data provided by the GPS, in addition to the developer adjustment to that value, because of the inaccuracy percentage of GPS in mobile devices.

5 AR Integration

There are several factors that have to be taken into consideration for outdoor rendering of augmented objects. One of these factors is lighting, which is a crucial component in rendering any object in a scene.

In order to improve the integration and create a realistic scene, AR platform should track sun position in real time, thus approximating lighting conditions of physical objects along with their shadows. Therefore a directional light is used to simulate sunrays.

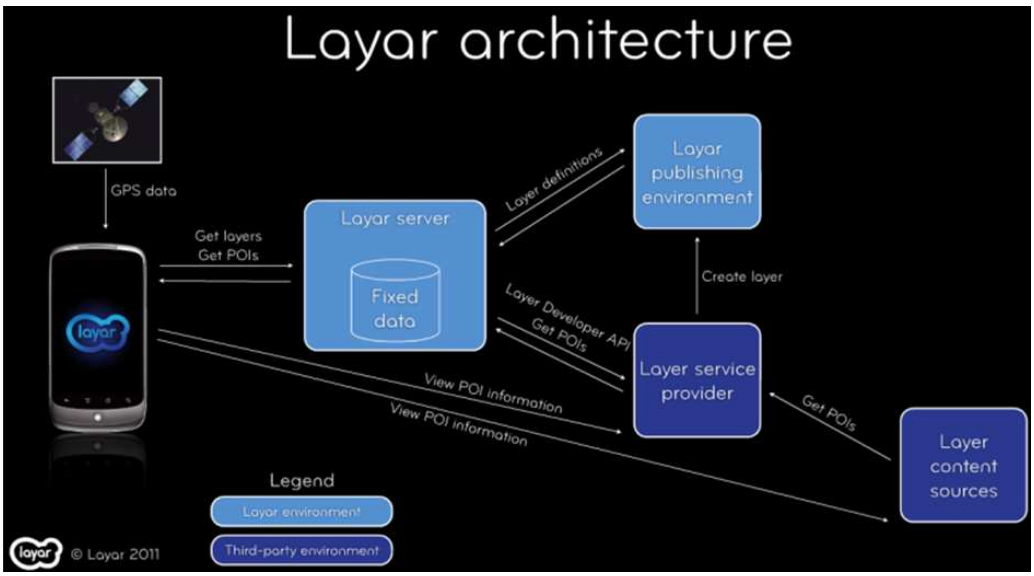


Figure 4: Layar Architecture as presented in Layar Developer Conference 2011 (image courtesy of Layar.com)

Earth is relatively spherical celestial object that rotates around itself eastwards every approximately 24 hours and around the Sun approximately every 365 days. The axis on which Earth rotates is the Polar axis. The great circles that intersect with the Polar axis are called meridians. The great circle equidistant from the North and South Pole is the equator [12]. Earth rotation axis is tilted by 23.4, which results in changing the relative position of the Sun as the Earth moves in orbit. This change reflects on the angle of the Sun rays according to the equatorial plane. This angle is called declination.

Figure 6 shows the Sun position towards the Earth along with the above described angles.

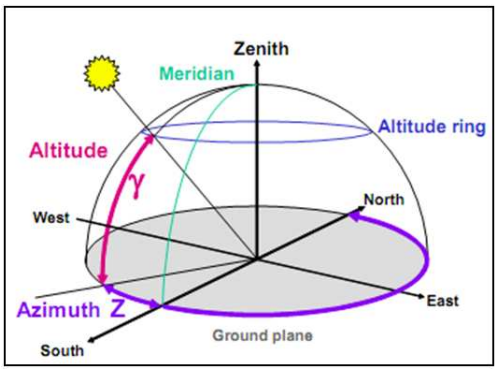


Figure 6: Sun position towards the Earth [12]

γ is the altitude of the Sun above the ground (horizon) plane and z is the azimuth, which is the compass direction of the Sun on the ground plane. Declination is calculated using equation 2:

$$Declination = 23.4 \times sin(\frac{360 \times (284+N)}{365}) degrees \quad (2)$$

Symbol	Variable	Definition
D	Declination	The angle of the Sun rays to the equatorial plane, positive in the summer.
L	Latitude	The angle from the equator to the position on Earth's surface
H	Hour angle	The angle the Earth needs to rotate to bring the meridian to noon. Each hour of time is equivalent to 15 deg.
N	Day number	The day number, January1stis 1.

Table 1: Azimuth Altitude equation legend

where N is the number of the day for which the declination is being calculated, January1st being day number 1. The Azimuth may be expressed in two ways: either as the angle clockwise from North or as the angle East of or West of South. Although the former is most often used, we used the latter convention. Azimuth and altitude of the sun can be calculated using the following equations [12]:

$$sin\gamma = (cosD \times cosL \times cosH) + (sinD \times sinL) \quad (3)$$

$$cosz = \frac{(cosD \times cosH \times sinL) - (sinD \times cosL)}{cos\gamma} \quad (4)$$

Table 5.1 shows the legend for the above equations:

The color of the Sun rays plays an important role in displaying the time of the day, and also determines the color temperature of objects. This problem can be ap-

Source	RGB(0-255)	RGB(0-1)
Sun at sunrise or sunset	182 126 91	0.71 0.49 0.36
Direct sun at noon	192 191 173	0.75 0.75 0.68
Sun through clouds/haze	189 190 192	0.74 0.75 0.75

Table 2: Sun color at different times of the day

proached by implementing atmospheric scattering algorithms to change sun and atmosphere's color depending on the time of the day. However in order to decrease the CPU/GPU load, and increase the frame rate, Sun color could be determined through basic hard coded RGB values [3] as shown in table 5.2:

Using those values, the RGB numbers between the two stages of the sun during the day could be extrapolated based on the starting time of the simulation.

Another factor that has to be taken into consideration is the night time, where no sun or any light source is available except the presence of the Moon, at a certain times of the month. In real life, the moonlight illumination is almost unnoticeable in urban or artificially illuminated areas, thus the moonlight factor will be neglected in this case (Figure 7).

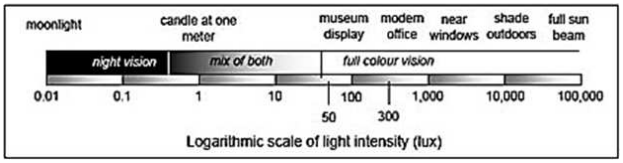


Figure 7: Logarithmic scale of light intensity (image courtesy of Canadian conversation institute)

In order to compensate the loss of Sun light, Rendering Engine component queries for artificial light that is associated with the desired model from database on the server side. The artificial light data contain position, light type, and the diffuse components. At the early dawn or late sunset, where Sun's illumination is not strong enough to illuminate the augmented object, the directional light is turned off, and the queried artificial lights are activated.

One possible way to speed up lighting calculations is by performing the light map, which is especially useful when used in conjunction with multi-texturing. Additionally, texture baking could be used as an alternative technique for increasing the frame rate [10].

6 Algorithm

In previous sections, registration, organization and integration techniques used in this AR platform were discussed. This section discusses the pseudo flow of events that are triggered in the AR platform, starting with user authentication and ending by augmenting the 3D model on the user's screen.

1. Authenticate user.

2. Get all GPS coordinates from server.

3. For every x period of seconds, check if user is at a close distance from a target area.

4. If user is close to a target, get permission to download the 3D object from user.

5. Create frustum based on the gravity vector extracted from the accelerometer and get rotation angle from compass.

6. Start calculating light position.

6.1 If it is day:

6.1.1 Create directional light resembling sun light.

6.1.2 Rotate light in .. axis for .... degrees.

6.1.3 Rotate light in .. axis for .... degrees.

6.1.4 Set directional light diffuse RGB values based on sun diffuse RGB lookup table.

6.2 If it is night:

6.2.1 Get artificial light data from server.

6.2.2 Create artificial lights based on their types.

7. Augment 3D object on the video stream.

When working with mobile devices, it is important to optimize mobile - server communication frequency. Therefore, after user authentication, all GPS coordinates are queried and stored in a list on the client side of the AR platform as shown in line 1. Thus it is not necessary to connect to the server unless the user permitted the client to download the target object as in line 4, or when it is night time and the client downloaded the artificial light data from the database as in line 6.2.1.

For every user-defined period of seconds, the application checks if the user is in the radius of the target location (i.e. line 3). When user confirmed to download the 3D object, the rest of the calculations are carried on the mobile device to minimize the communication frequency with the server.

Sun rotations around x and y axis are calculated using formulas 3 and 4.The altitude and azimuth of the frustum are derived from the accelerometer and compass data. Finally after all calculations are done, the object is rendered on mobile device's screen, and the frustum is updated based on the GPS, accelerometer and compass data.

7 Results

Figure 8: Layar Augmented 3D object at noon (image courtesy of Layar.com)

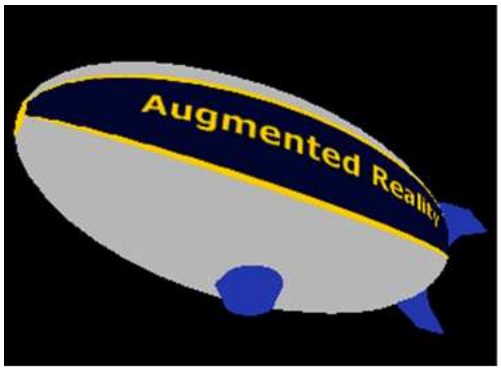


Figure 9: Default OpenGL Light (i.e. OpenGL lights not enabled)

Figure 10: Direct Sun at noon (image courtesy of Layar.com)

By the time of writing this paper, tests have been running under iPhone emulator, and since computers video cards can handle OpenGLES2.0 with almost no frame rate loss, as a result, frame rate data will be neglected for the time being. Current models are using per-vertex illumination, and the shaders component of this platform is still under development. As a consequence, directional light and spotlight with insignificant diffuse values for smoothing shadows have been added. However, the loss of per-pixel illumination does not highly affect the outcome; an image displaying the color temperature of the models at different times of the day is shown in Figure 8 compared with Layar render for the augmented 3D zeppelin model:

Initially, comparing Figure 8 with Figure 9, that Layar AR Platform did not include any lights for illuminating its 3D objects. Nevertheless we could see a dramatic change even on per-vertex lighting when comparing 10 and 11 renders. It is worth noting that comparing Figure 10 and 11 carefully, the latter has lighting temperature close to the camera feed and almost has the same sunray direction. Color temperature of objects during noon could have a bit of yellow tint to it, as shown in Figure 10 on the building bellow the zeppelin and the test zeppelin render (Figure 11). Figure 12 shows the zeppelin model under sunset lighting condition, while for the cloudy or hazy weather as showed in 14 light color tends to be white. We believe that, completing the shader component we will gain better results. After several tests, one could disagree with the

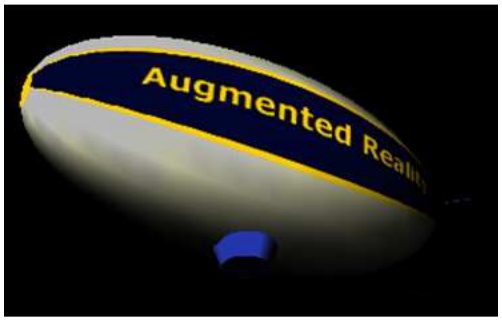


Figure 11: Test render direct sun at noon

Figure 12: Sunset

approach proposed by Birn for using same values for sunrise and sunset [3]. The reason is that during sunrise (i.e. dawn), objects' color temperature tends to be cold, thus has blue tint to it as shown in Figure 13. Therefore, one could suggest adding an additional RGB value for sunrise. We suggest that these values would be (0.50 0.49 0.60) for R, G and B, assuming that RGB values goes from 0 to 1.

8 Conclusions and Future Work

In this paper a solution for AR organization and integration problems, in a context of generic AR platform is presented. This solution deals with two major topics: integration where virtual objects are illuminated according to the time of the day, by tracking sun position in real time, thus estimating the correct color temperature and shadows; and organization where all objects are organized in a relational SQL database along with its illumination data on remote server. 3D objects are downloaded upon a query from the client side (i.e. mobile device), hence superimposing the downloaded object after light calculations are finished.

The presented platform is still under development, though several features could be implemented, such as calculating length of shadows, compensating for rainy, snowy or cloudy weather, where most of the objects tend to have no shadows since light distortion is very high and the object is illuminated almost from all sides. Implementing several lighting techniques, such as light mapping or texture baking could assist in increasing the frame rate.

Expanding database to include the ability to subscribe to

Figure 13: Sun at Dawn

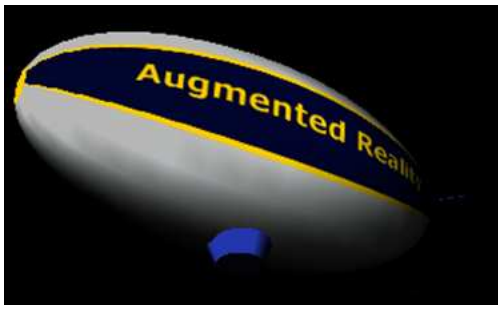


Figure 14: Cloudy/Hazy weather

the web server could encourage developers and/or clients to upload their own content. Furthermore, database could be expanded to contain other types of multimedia, such as audio, that could be associated to a certain 3D model.

Other improvements to this AR platform would include giving the ability to developers and content providers to add to and enhance the platform. Current calculations for predicting Sun position still lack a very important parameter, which describes the accurate position of the user on Earth. Therefore Sun position would be calculated with a higher precision.

Another important factor that could affect the experience of the users is the quality of the camera in their mobile device. Since modern mobile devices are equipped with autofocus and/or have automatic exposure correction, one possibly would notice the difference of color temperatures and light intensity between the augmented object and the physical world. This platform could not fully enhance the integration if augmented objects exist in urban areas, as when the augmented object's position has special geological properties, or it is surrounded by other higher objects that cast shadows on it.

References

[1] A hybrid registration method for outdoor augmented reality. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, pages 67–, Washington, DC, USA, 2001. IEEE Computer Society.

[2] Ronald T. Azuma. The challenge of making augmented reality work outdoors. In *In Mixed Reality: Merging Real and Virtual*, pages 379–390. Springer-Verlag, 1999.

[3] J. Birn. *Digital Lighting & Rendering*. New Riders, 2006.

[4] S. Cawood and M. Fiala. *Augmented Reality: A Practical guide. The Pragmatic Programmers*. 2007.

[5] Lin Chai, William A. Hoff, William A. Hoff (corresponding, and Tyrone Vincent. 3-d motion and structure estimation using inertial sensors and computer vision for augmented reality. *Presence*, 11:474–492, 2000.

[6] Steven Feiner, Blair MacIntyre, Tobias Hllerer, and Anthony Webster. A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. *Personal and Ubiquitous Computing*, 1:208–217, 1997. 10.1007/BF01682023.

[7] Masayuki Kanbara and Naokazu Yokoya. Real-time estimation of light source environment for photorealistic augmented reality. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 2 - Volume 02*, ICPR '04, pages 911–914, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Saulo A. Pessoa, Eduardo L. Apolinario, Guilherme de S. Moura, Joao Paulo S. do M. Lima, Marcio A. S. Bueno, Veronica Teichrieb, and Judith Kelner. Illumination techniques for photorealistic rendering in augmented reality. In *SVR2008*, 2008.

[9] Gerhard Reitmayr and Dieter Schmalstieg. Location based applications for mobile augmented reality, 2003.

[10] P. Ridout. *iPhone 3D Programming*. O-REILLY, 2010.

[11] Reiner Wichert. A mobile augmented reality environment for collaborative learning and training. In Margaret Driscoll and Thomas C. Reeves, editors, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2002*, pages 2386–2389, Montreal, Canada, 2002. AACE.

[12] M. Wilkinson. *Building Enironment 1*. University of Bath.

[13] www.layar.com.

[14] Suya You, Ulrich Neumann, and Ronald Azuma. Orientation tracking for outdoor augmented reality registration. *IEEE Computer Graphics and Applications*, 19:36–42, 1999.

Modern Methods of Realistic Lighting in Real Time

István Szentandrási
Supervised by: Adam Herout

Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

Physically plausible illumination in real-time is often achieved using approximations. Recent methods approximate global illumination in the screen space by exploiting the capabilities of modern graphics cards. Two of these techniques, screen-space ambient occlusion and screen-space directional occlusion, are described in this work. Screen-space directional occlusion is a generalized version of screen-space ambient occlusion. It supports one indirect bounce of diffuse light and depends on the direction of incoming light. The main goal of this project is to further experiment with these methods and improve them. For a uniform distribution of the sampling points, the Halton sequence is used. In order to reduce the noise, geometry-aware bilateral filtering is presented. Methods are further sped up by computing them in a lower resolution, and they are restored to full resolution using joint bilateral upsampling in order to create the final image.

Keywords: global illumination, ambient occlusion, screen-space ambient occlusion, screen-space directional occlusion, halton sequence, bilateral filtering

1 Introduction

Computing global illumination in real-time has been and still is a major challenge in computer graphics. Due to the complexity of light transport and some material properties, real-time frame rates can only be achieved at the cost of trade-offs and rough approximations. Perceptually among the most important optical phenomena belong soft shadows and indirect lighting. There have been many attempts to simulate either of these in real time. A handful of these attempts are based on ambient occlusion (AO) [13], which is very popular in the film industry as well as in games. The main advantage of these techniques lies in their speed and simple implementation.

As in every approximation, ambient occlusion has some limitations, too. The basic method [13] displays darkening of cavities; however, it does not take into account the direction and intensity of light coming from light sources or environmental maps. A better method has been introduced by Ritschell et al. [11] called screen-space directional occlusion (SSDO). SSDO provides more realistic illumination: it accounts for the direction of the incoming light and supports a single indirect bounce of light.

The aim of this work is to experiment with these methods and improve them. The improvements are mostly focused around speeding up the methods. This work is structured as follows. In Section 2 we describe the present state of methods used in the area. Then Section 3 we will present screen-space ambient occlusion (SSAO) and SSDO.

Section 4 presents possible modifications and optimizations to the SSDO method: speed optimizations, an alternative method for uniform distribution of sampling points using Halton sequence and using a variable number of sampling points for each pixel based on local scene complexity. Section 5 summarizes the achieved results.

2 Related Work

Ray tracing and radiosity have always been the two basic approaches to approximate physically correct lighting. However, both methods require massive computations. In the case of radiosity, it solves a system of equations. In the case of ray tracing, the major slowing factors are the number of object-ray intersections and visibility tests. There are many techniques to accelerate these methods, such as final gathering, irradiance caching, sparse sampling, advanced space division structures, etc. Even with the recent growth in processor speeds and the introduction of GPGPU solutions, these methods are still too slow for interactive applications. Rendering on GPU remains the superior solution for real-time rendering. It still has a major lead, especially for dynamic scenes. This barrier caused the development of alternative methods which did not try to simulate physically correct lighting. They just aim to give perceptually convincing approximations.

Since the introduction of ambient occlusion [2][13], it has been widely adopted both in gaming and the film industry. Ambient occlusion computes the visibility of the hemisphere at each point of the scene. The method is often calculated by casting rays in every direction over the hemisphere using Monte Carlo sampling. The calculated factor is used to modulate ambient lighting, just as the name suggests. [4]

Casting rays in every point still requires too much computing power, so a few alternative methods were introduced. These methods compute AO less accurately in order to achieve higher frame-rates. Instead of computing occlusion over surfaces in 3D, these methods usually approximate AO in the screen space [12][8][1][5]. SSAO is very popular due to its simplicity and speed. It does not require any additional data and can be applied as a post-process to the scene.

Ambient occlusion is just a rough approximation of general light transport. It does not take into account any directional information or other more expensive illumination effects (interreflections, caustics, subsurface scattering). A different family of techniques, the precomputed radiance transfer (PRT) [4], does support the aforementioned features. On the other hand, PRT algorithms typically assume static scenes, distant lights or environment maps.

Screen-space directional occlusion (SSDO) [11] tries to combine the speed and simplicity of SSAO methods with directional information of lighting and near field indirect color bleeding.

In order to avoid some limitations of screen space ambient occlusion, a hybrid method was introduced by Reinbothe et al. [10]. This method works in 3D space by voxelization of the scene, calculating occlusion based on this information and finally using bilateral filtering in the screen space to smooth the shadows.

A completely different approach was taken by Kaplanyan et al. [6]. They approximate indirect illumination in fully dynamic scenes using cascaded light propagation volumes. This method supports single bounce illumination with occlusion, but it can be extended to support multiple bounces and to handle participating media.

These techniques show that in order to generate visually convincing images, no physically precise computations are needed. Simple approximations using soft shadows, ambient occlusion, and optionally, a single bounce of indirect light can give out acceptable results even in real-time.

3 Real-Time Global Illumination Techniques

There are many techniques to approximate global illumination in real time. We focused on methods that can be computed in a postprocessing step so as to improve overall quality of the rendered images.

3.1 Screen-Space Ambient Occlusion

Screen-space ambient occlusion (SSAO) is a coarse approximation of ambient occlusion that works in the screen space in order to achieve real-time frame rates. The idea behind SSAO is to reuse the z-buffer data, which was already computed during the rendering of the scene. This approach is based on sampling the surrounding pixels combined with simple depth comparisons. Based on these results, an average visibility value can be computed. This visibility property is used as a darkening factor for cavities and corners in the scene. This way SSAO can be computed in one pass as a post-process over the image.

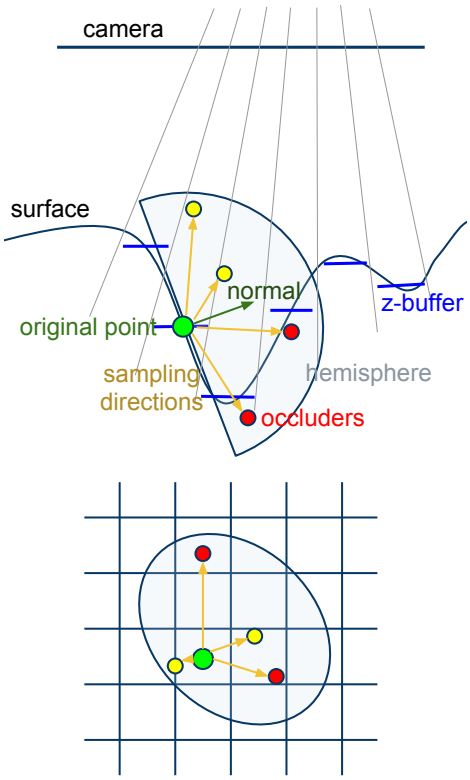


Figure 1: SSAO principle. An area around the pixel is sampled in 2D (bottom image). Based on the normal, sampling points are generated inside the hemisphere (top image). The generated depths for each sampling point are compared to the depth of the appropriate pixel to determine if the pixel corresponds to an occluder object (yellow (light) and red (dark) points. Based on the results of the depth comparisons, the pixel's intensity will be darkened (on the image to half, since half of the sampling points are below scene surface).

Additional information can be used to achieve more precision and hence better looking results. The normals for each pixel could be a good choice. Without taking the normal into account and just randomly generating points in a close proximity of the pixel may cause darkening of unwanted parts of the scene, such as planes that are almost parallel with the view direction. Using the normal of the pixel sampling directions outside the hemisphere can be filtered out (Figure 1). However, random sampling may cause problems in the image which have to be sorted out. So as to avoid artifacts, the distribution of the generated random samples over the hemisphere is needed to be as uniform as possible. Also, the noise caused by non-uniform sampling should be smoothed.

Usually in SSAO methods, the problem of generating

samples randomly with a uniform distribution is solved by precomputing a few uniformly distributed random directions in a sphere. Variation to the sampling points is added using random normals to reflect the directions. These new directions are then optionally reversed to be in the hemisphere around the normal of the pixel. In this project we used the Halton sequences with appropriately chosen bases.

Computing ambient occlusion from simple depth values has some drawbacks, too. First the precision of SSAO is highly dependent on the size of the scene. The bigger the scene, the coarser the object shape approximation. This can lead to unwanted effects, like darkening the whole silhouette of an object.

The second problem is caused by the limited area that is sampled for occluders. Let us consider two objects close to each other in the scene in 3D. Using a classical ambient occlusion method, such as using ray tracing, the two objects are darkened. However, using SSAO the distance between the projected positions of the objects might be larger then the sampled area. As a consequence SSAO will not detect any occluders and the objects will not be darkened.

3.2 Screen-Space Directional Occlusion

SSDO is a fast approximation of global illumination. It works in the screen space, takes into account the direction of the light, and is able to handle one indirect bounce of diffuse light [11]. In order to compute light transport, SSDO uses the 3D positions and normals of each pixel in the screen space as input. The output is created in two passes. In the first pass, the direct illumination is computed. In the second pass, the indirect bounce of light is computed using the data from the previous pass.

3.2.1 Direct Illumination Using Directional Occlusion

While standard SSAO methods use only the positions and normals of each pixel, SSDO also takes into account the direction of the incoming light. The amount of directional light is computed as follows for each point $\mathbf{P}$ and normal $\mathbf{N}$:

$$L_{dir}(\mathbf{P}) = \frac{1}{\pi}\int_{\Omega}\frac{\rho}{\pi}L_{in}(\omega)V(\omega)(\mathbf{N}\cdot\omega)\,\mathrm{d}\omega, \qquad (1)$$

where $\frac{\rho}{\pi}$ is the diffuse BRDF, L_{in} is the incoming radiance from direction ω in the hemisphere Ω and V is the visibility test. When using Monte Carlo sampling the integral is replaced by a sum of K samples each covering a solid angle of $\Delta\omega = 2\pi/K$:

$$L_{dir}(\mathbf{P}) = \sum_{i=1}^{K}\frac{\rho}{\pi}L_{in}(\omega_i)V(\omega_i)(\mathbf{N}\cdot\omega_i)\Delta\omega. \qquad (2)$$

This method assumes that L_{in} can be efficiently computed from environment maps or point lights. Similar to SSAO, so as to avoid ray-tracing, the occluders are approximated in the screen space. The difference is that while in SSAO the samples are generated in 2D in image space, SSDO uses sample points generated in the hemisphere in 3D using the normal and the 3D position of the pixel. The sample points are then backprojected into the image space in order to determine if within the given direction is an occluder or not. This way SSDO does not suffer from the problem mentioned in SSAO, that is: objects further away in the screen-space, but closer in 3D in the scene may cause occlusion.

The sampling of the hemisphere is done in the following way: for every generated direction ω_i and a random step $r_i \in [0..\, r_{max}]$, the position of the sampling points is computed as $\mathbf{P} + r_i\omega_i$. The generated points are located in the hemisphere with a center of $\mathbf{P}$ and oriented around the normal $\mathbf{N}$. The depth of the backprojected sampling points is compared with the values from the original z-buffer. If the depth value from the original z-buffer is smaller than the depth of the sampling point, the sampling point is below the surface. The light from this direction is blocked by an occluder. Otherwise, the light has a clear path from this direction and the incoming radiance can be computed.

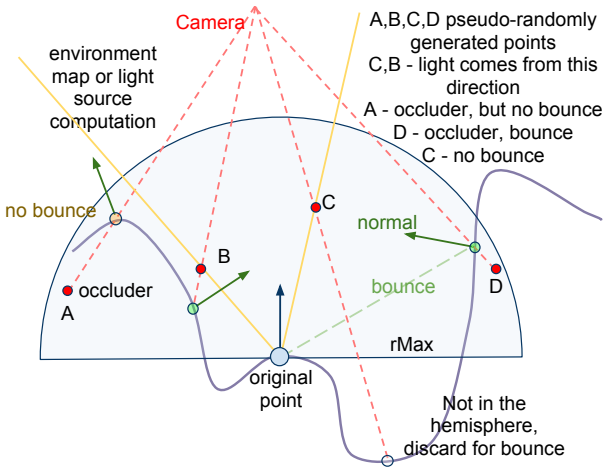


Figure 2: SSDO principle. Random samples are generated in 3D in the hemisphere. Samples under the surface are classified as occluders. Otherwise, the incoming radiance can be computed from the direction defined by the sampling point. The sampling points classified as occluders are projected on the surface. Based on the color and position of the pixels on the surface, indirect bounces are computed.

The method is demonstrated in Figure 2 for four sampling points A, B, C, D (red dots). The sampling points are generated randomly with a uniform distribution over the hemisphere with a random step from the original point and then backprojected onto the image. Now that the image space coordinates are known, the 3D coordinates can be computed or read from a frame buffer (green, orange and black dots). These points are again projected into the image in order to get the distance from the camera. If the sampling point is further than the appropriate point on a surface in the scene, the sampling point is classified as an

occluder (A and D points). Otherwise, the illumination can be computed from the direction defined by the point and the origin (B and C points). The direction is shown by the yellow line.

3.2.2 Indirect Bounce

Since the 3D position and normal are available for each sampling point projected to the surface, they can be used to get one indirect bounce of light from the given directions. In the original paper, only the points on the surface projected from sampling points classified as occluders were taken into account (A and D points). For each of these pixels the computed directional light intensity and the corresponding pixel color from the direct illumination pass is used as the base for indirect light. In order to calculate the indirect radiance sent to the origin, these pixels are treated as small patches oriented around the normal. Using the sender normal, back facing patches can be filtered out (for example, for sampling point A).

The equation for the additional incoming indirect radiance for a point $\mathbf{P}$:

$$L_{ind}(\mathbf{P}) = \frac{1}{\pi}\int_{\Omega}\frac{\rho}{\pi}L_{dir}(\mathbf{P}_{\omega})(1-V(\omega))\cdot \frac{A_s(\mathbf{N}\cdot\omega)(\mathbf{N}_{\omega}\cdot(-\omega))}{|\mathbf{P}-\mathbf{P}_{\omega}|^2}\,\mathrm{d}\omega, \quad (3)$$

where $\mathbf{P}_{\omega}$ and $\mathbf{N}_{\omega}$ are the point and normal from point and normal buffer, each corresponding to a sampling point taken from the hemisphere in direction ω. A_s is the area associated with the sender patch. This equation respects the mutual orientation of the surfaces based on the normals $((\mathbf{N}\cdot\omega)(\mathbf{N}_{\omega}\cdot(-\omega)))$ and that the intensity of the incoming radiance decreases quadratically with the distance of the surfaces.

The modified version of the equations for K samples is then:

$$L_{ind}(\mathbf{P}) = \sum_{i=1}^{K}\frac{\rho}{\pi}L_i(1-V(\omega_i))\frac{A_s(\mathbf{N}\cdot\omega_i)(\mathbf{N}_i\cdot(-\omega_i))}{|\mathbf{P}-\mathbf{P}_i|^2}\Delta\omega, \quad (4)$$

For the initial value for A_s, the base circle is subdivided into K regions, each covering $A_s = \pi r_{max}^2/K$. This value can also be used as a parameter to control the strength of the color bleeding manually.

In the example above (Figure 2), points A and D are the only occluders. After projecting these points onto the surface from the cameras viewpoint, the information about the normal, position and color are also available. The projected point for A has a back facing normal, so it will not contribute to the final bounce. The patch for sampling point D, on the other hand, will qualify as a sender of indirect light towards point $\mathbf{P}$.

4 Modifications and Implementation Notes

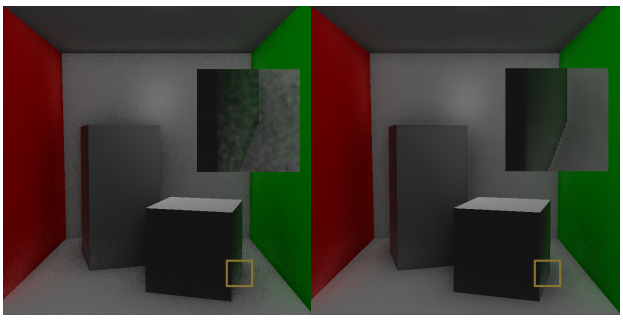
The aim of this project is to experiment with the methods described in the previous chapter and to potentially improve them. We will next describe three modifications and improvements we experimented with and which are potentially beneficial.

4.1 SSDO for Directional and Point Lights

When SSDO is computed for scenes with directional and point lights, several modifications are possible. Computing the pixels intensity in real implementations based on the equation (2), due to the random or pseudo-random sampling of the directions in the hemisphere, causes noise even on plain surfaces with no occluders nearby. So as to avoid this, SSDO can be computed as follows: when rendering the scene before computing SSDO to get the material, normal, depth information, the overall intensity of the pixels can be computed using the appropriate lighting model. The overall intensity should be equal to the value computed using equation (1) without the visibility function. When computing direct illumination (first step) in SSDO, instead of computing the intensity for the pixels, a modulation factor can be computed for each pixel. This modulation factor is the ratio of the intensity computed with the visibility function and without it using equation (2). The modulation factor can be used to darken the diffuse and/or the ambient component of the overall intensity when creating the final image. The modulation factor can either be per channel or a single value based on scene and lighting properties. When there are multiple lights in the scene with radically different colors, a darkening factor for each color channel is best. However, for most scenes a single modulation factor is sufficient, since the lights are usually white or the distance between them is large enough, so they have only a minor effect on objects close to other lights. A single modulation factor also helps to reduce memory usage.

This modulation factor can be further processed. In order to reduce noise, smoothing can be used. A simple Gaussian blur is not enough in this case. The usage of geometric information is necessary to prevent bleeding of values over edges and between distant pixels in 3D, but close in 2D. A geometry-aware filtering, like a modified bilateral filtering on the base of normals and depth value, is suitable. Firstly, this method was used by Reinbothe et al. [10]. So as to approximate the results of a full bilateral filtering, they separated the calculations into a vertical and horizontal pass. A combination of the results of these two one-dimensional filters improves the frame rates significantly and still provides an acceptable quality.

Since this modulation factor is actually a generalized version of the darkening factor computed with ambient occlusion, it changes, similar to the ambient occlusion fac-



(a) No filtering, 10 sampling points - 74FPS

(b) 2x subsampling (3x3 upsampling kernel, 20 sampling points per pixel and 11x11 bilateral filtering - 78 FPS

Figure 3: Comparison of non-filtered and smoothed results. Regions with the yellow border are shown in more detail on the right side of the images.

tor, slowly over spatial space on surfaces. This permits one to compute it in lower resolutions. As a consequence the method could be sped up radically; on the other side, the result should be upsampled correctly. For this joint bilateral upsampling [7] can be used with the same modifications as for smoothing to honor geometry properties (Figure 3).

More speed improvement can be achieved by merging the two steps of SSDO together. This means that the source color for the bounces is taken from the scene rendered without SSDO darkening (the same image, that is later darkened by SSDO). The modulation factor and indirect bounces are stored and filtered separately. In order to avoid pixels being too bright in darkened corners, the indirect bounces should also be darkened using the modulation factor.

4.2 Sampling and the Halton Sequence

In both SSAO and SSDO, Monte Carlo sampling is used to approximate the correct solutions. Absolute random distribution of samples in Monte Carlo methods can cause problems; one of the worst of these is clumping (when for small number of random numbers, the variance between the values is low). So as to decrease the effect of clumping in samples, quasi-Monte Carlo methods eliminate the randomness completely. Samples are deterministically computed to achieve a stochastic distribution as close to the uniform distribution as possible.

In order to describe how much the point distribution of a given method derivates from an ideal solution, a measure called discrepancy is used. Quasi-Monte Carlo methods try to minimize this discrepancy. There are several low-discrepancy sequences that are used for generating sampling points: Hammersley, Halton, Sobol, Niederreiter, etc. [4]

The Halton sequence generation is based on the radical inverse function applied to an integer i. This integer can be expressed in a base b with terms a_j:

$$i = \sum_{j=0}^{\infty} a_j(i) b^j. \tag{5}$$

The radical inverse function is computed by reflecting the resulting digit sequence around the decimal point:

$$\Phi_b(i) = \sum_{j=0}^{\infty} a_j(i) b^{-j-1}. \tag{6}$$

For generating multi-dimensional low-discrepancy sequences, a different radical-inverse sequence is used in each dimension. The ith point in the sequence is given as:

$$x_i = (\Phi_{b_1}(i), \Phi_{b_2}(i), \ldots, \Phi_{b_d}(i)), \tag{7}$$

where the bases b_j are relatively prime and d is the dimension of the sequence.

An intuitive explanation of the uniformness of the Halton sequence is as follows. Let us consider the generated floating point numbers as digit sequences in the given base expressed as strings. Before generating strings of length $m+1$, all the strings of length m are produced. This means that before generating a new point on an interval, all intervals of size $b-m$ will be visited first. This fact also suggests some kind of periodicity in similarity of the generated values. For example, let us consider a Halton sequence with base 2: $\Phi_2(50) = 0.296875, \Phi_2(50+16) = 0.2578125, \Phi_2(50+32) = 0.2890625, \Phi_2(50+64) = 0.3046875$. This periodicity can be expressed [3]:

$$\left|(\Phi_b(i) - \Phi_b(i+mN_g)\right| < \frac{1}{b^k}; N_g = lb^k, l > 0, k \geq 0, \tag{8}$$

where l, m, k are integers, and N_g is the period to generate similar sampling points. For multidimensional Halton sequences, the least common multiple of the periods for each dimension is used as N_g. This property is demonstrated in Figure 4.

This periodicity can be exploited to control the number of sampling points and still have a quasi uniform distribution. For example, with a period 10, 10 samples can be used for pixels where the low number of sampling points does not matter. For pixels, where more sampling points are needed to get a smoother result, $20, 30, \ldots, k \cdot 10; k \in N$ sampling points can be used (see Section 4.3). Another possibility for future improvements may be, for example, in the case of occlusion detection to filter out directions, where an occluder was already found.

4.3 Preprocessing

In SSDO, the same amount of sampling points is generated for every pixel. Setting this amount higher means better results, but it will cause a performance drop. Generating more samples for planes which do not have any

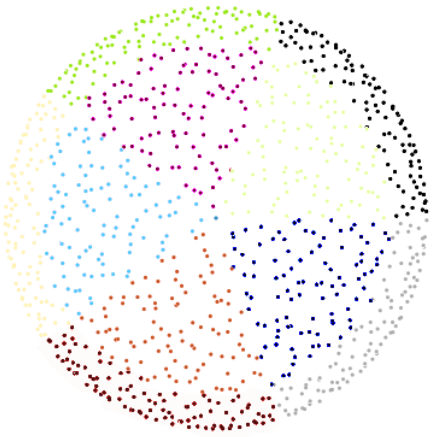


Figure 4: Projected view of the hemisphere - Halton sequences with bases 2, 5 and period 10. Each pixel is colored based on the index in the sequence and the period. Every 10th pixel has the same color.

occluders nearby is a waste of computing power. The solution would be to generate more sampling points for pixels which are potentially occluded and less for pixels where the probability of finding an occluder is low. The number of occluders is usually higher at parts of the image where the normals or the depth values differ significantly. So as to get areas where the higher number of sampling points should be generated, a preprocessing step could be added to SSDO calculations. In this preprocessing step, a weight is calculated to define the number of sampling points to be used. Areas where the changes in normal and depth values are bigger are given a higher weight. Planes, on the other hand, will have much smaller weight since the normals are constant for pixels on the same plane.

Figure 5: Preprocessed images with different sizes of filtered images. The darker regions represent higher number of sampling points. The blue color just shows the computed SSDO darkening factor.

In order to calculate this weight, a simple filtering using the normal and depth values can be used to get the gradient magnitude. This value can be used to determine the number of sampling points. So as to get wider areas around discontinuities and in order to speed up the filtering, it can be computed in much lower resolutions. Optionally combining results from different resolutions should give more precise results of areas where SSDO values may change more. Using just one resolution is not the best solution. A more advanced method was used by Nichols et al. [9] to get locations where higher resolution was required for computing image space radiosity.

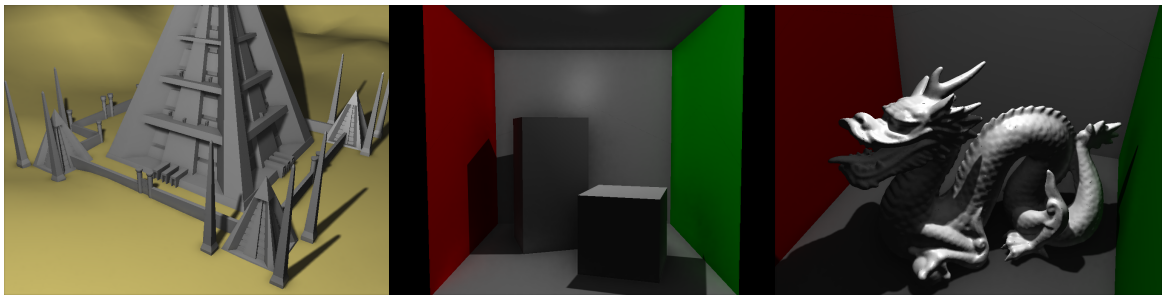
5 Results

With both methods (SSAO and SSDO) we were able to produce realistic images in real-time (Figure 7). The results of the achieved frame rates are summarized in Table 1. The methods were tested on three scenes with resolution 1024x768 (Figure 6). These numbers are just exemplary. The speed of these methods depends also to a great extent on the resolution, graphics hardware, as well as the degree of required smoothing.

SSAO was naturally the fastest. The additional computation and texture reads to get bounces for SSDO makes it 25% slower with the same number of sampling points. However, to get good results for SSDO with a larger hemisphere radius, many sampling directions are needed. From the measured frame-rates it is clear that the limiting factor for SSDO computation is the speed of the fragment shader and the number of texture lookups per pixel. These are the areas that should be more optimized in the future.

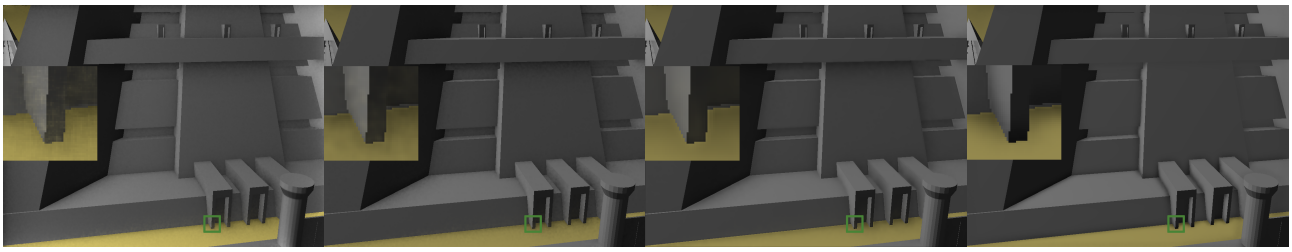
Due to the fact that SSDO is computed in the screen space, a few problems arise. The lack of complete knowledge of 3D causes occluders not visible from the camera's point of view to be discarded, hence making the results highly view-dependent. For more complex scenes even a small change in camera position may reveal parts of the scene previously hidden and cause new shadows and indirect bounces. The authors of the original paper suggested using depth peeling for partly solving this problem. Storing multiple depth values for each point in multiple passes gives more information on the scene structure; however, it makes the speed of the technique dependent on scene complexity. It also further slows down occlusion computations by forcing it to read values from multiple buffers and calculating the depth test for each. Alternatively, the authors also suggest using multiple viewpoints. This in theory could give better results than depth peeling, but correct positioning of the cameras may vary based on the scene type.

A comparison of SSAO and SSDO can be seen in Figure 8. The difference in the two techniques can be easily observed on the darkening factor around the tail of the dragon. While SSAO darkens only the silhouette, the tail in SSDO darkens the wall close to it in 3D. The next difference is when the light position is changed. While SSAO remains static with moving light, the shadows caused by occlusion in SSDO move a little in the opposite direction. This nice feature of SSDO with the addition of the bounce gives us much more believable results than SSAO.



(a) Pyramids (b) Cornell box (c) Dragon

Figure 6: Scenes rendered using SSDO with variable number of sampling points based on preprocessing (20, 30, 40 or 50), 2x subsampling, 11x11 kernel for bilateral smoothing and 3x3 upsampling filter. (Dragon model is from the The Stanford 3D Scanning Repository)



(a) SSDO 2 step (b) SSDO 10 points (c) SSDO with preprocessing and subsampling (d) SSAO

Figure 7: SSDO 2 step – separate step for computing direct and indirect illumination; SSDO 10 points – SSDO computed in one step using modulation factor; SSDO with preprocessing and subsampling – SSDO computed in one step with 2x subsampling, 3x3 kernel for upsampling and variable number of sampling points per pixel based on a preprocessing step (20, 30, 40 or 50 samples). For all methods 11x11 smoothing kernel was used. Regions with the green border are shown in more detail on the left side of the images.

[FPS]	Number of sampling points	Pyramids (15804 faces)	Cornell box (30 faces)	Dragon (201037 faces)
SSAO no subsampling	10	72	77	72
SSDO 2 steps	10	42	37	33
SSDO no subsampling	10	51	46	42
SSAO	20	115	127	113
SSDO	20	90.5	83	74
SSDO	50	67	55	49
SSDO with preprocessing	20-50	74	67	60

Table 1: Frame rates for each scene and method with 11x11 kernel for bilateral smoothing and 3x3 kernel for upsampling, if not specified otherwise. In the second column are the number of sampling points used per pixel. For the 'SSDO with preprocessing' row variable number of sampling points (20, 30, 40 or 50) was used based on a preprocessing step. For the rest of the rows constant number of sampling points was used. The 'SSDO 2 steps' row represents the original two-step algorithm with a separate step for computing direct and indirect illumination. For the other SSDO rows indirect illumination and modulation factor was computed in one pass. For testing an ATI Radeon™ HD 4850 GPU was used.

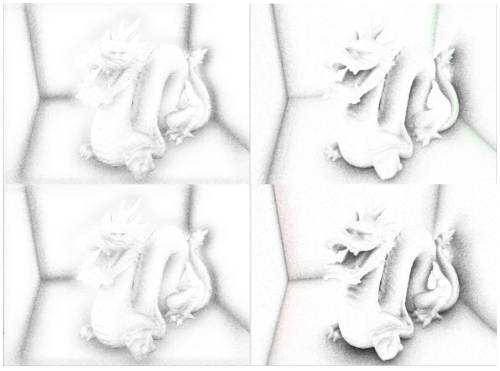


Figure 8: On the left SSAO; on the right SSDO with different light positions for the rows. The light is placed in front of the dragon, closer to the left wall in the top row of images and behind the dragon, closer to the right wall for the bottom row images.

6 Conclusions

Screen-space ambient occlusion is a very fast approximation of ambient occlusion, but it has some limitations. Screen-space directional occlusion includes two generalizations that add directional occlusion and diffuse indirect bounces. Both extensions improve realism considerably for a minor computational cost.

In this paper, a few experiments were made to the screen-space directional occlusion. Sample point generation based on the Halton sequence is an easy way to get uniform distribution of the sampling points. The periodic properties of the Halton sequence can also be used to potentially further optimize the method. This was exploited to generate a variable amount of sampling points, but still have a pseudo-uniform distribution. So as to reduce noise, a modified version of bilateral filtering was used, which took into account the geometry information as well to avoid color bleeding over edges and between objects. The SSAO and SSDO methods were computed in lower resolutions to speed up the method. For upsampling to the original resolution, joint bilateral upsampling was used to honor geometry properties.

In the future, more experiments can be made to SSAO and SSDO accompanied with more comprehensive testing based on the controllable features of each method. In order to get a clearer picture where these methods stand performance-wise, a few other methods could be explored, too.

References

[1] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH '08, page 22:1, New York, NY, USA, 2008. ACM.

[2] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *SIGGRAPH Comput. Graph.*, 15:307–316, August 1981.

[3] Kirill Dmitriev, Stefan Brabec, Karol Myszkowski, and Hans-Peter Seidel. Interactive global illumination using selective photon tracing. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 25–36, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[4] P. Dutré, K. Bala, and P. Bekaert. *Advanced global illumination*. Ak Peters Series. AK Peters, 2006.

[5] Dominic Filion and Rob McNaughton. Effects & techniques. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 133–164, New York, NY, USA, 2008. ACM.

[6] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 99–107, New York, NY, USA, 2010. ACM.

[7] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[8] Martin Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[9] G. Nichols, J. Shopf, and C. Wyman. Hierarchical image-space radiosity for interactive global illumination. page 11411149, 2009.

[10] C. Reinbothe, T. Boubekeur, and M. Alexa. Hybrid ambient occlusion. EUROGRAPHICS 2009 Areas Papers, 2009.

[11] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 75–82, New York, NY, USA, 2009. ACM.

[12] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 73–80, New York, NY, USA, 2007. ACM.

[13] S. Zhukov, A. Inoes, and G. Kronin. An ambient light illumination model. In *proceedings of the Eurographics Workshop in Vienna, Austria*, Rendering Techniques '98, pages 45–56, Springer, 1998.

Bidirectional Photon Mapping

Jiří Vorba
Supervised by: Jaroslav Křivánek

Charles University, Prague

Abstract

This paper introduces a method for optimal combination of light paths generated from the camera and from the light sources in the photon mapping algorithm used for computing global illumination. Our method is based on Multiple Importance Sampling, a general approach, introduced by Veach, for adaptive path connection in bi-directional path-tracing. Our goal is to examine this method in connection with the biased algorithm of photon mapping and to improve the ineffective final gather heuristic used in the original version of this algorithm. This heuristic is usually problematic when applied to the scenes where highly glossy materials prevail.

1 Introduction

Bi-directional methods for computing global illumination generate light paths both from light sources and from camera and afterwards they connect them together. A provably good strategy for connecting light paths introduced by Veach [1] is known for bi-directional path tracing (BDPT) [2]. However, the algorithm is not put to use that often in practice because of the slow convergence of some parts of the light transport.

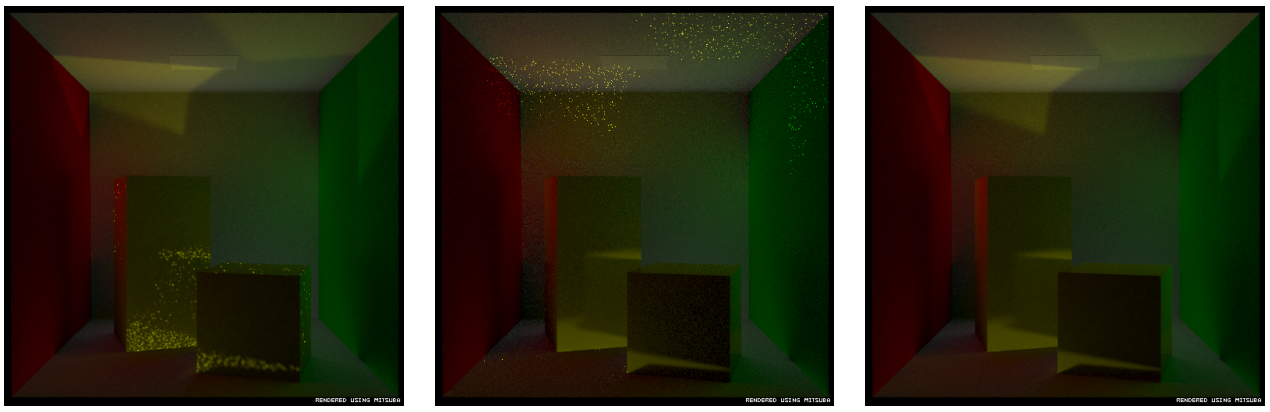
On the other hand, for popular bi-directional method like photon mapping [3] the optimal strategy for path connection is not known. The consequence is poor image quality in scenes containing many glossy materials. There are two reasons why objectionable artifacts usually occur when using photon mapping on glossy objects. First, radiance estimate on highly glossy materials suffers from high variance. Second, distribution rays cast during final gathering will usually hit the scene too close to each other because of the narrow BRDF peak. This results in highly correlated radiance estimates and the desired error averaging of rough information in the photon map is not achieved. Both of these issues result in the objectionable artifacts especially visible in the corners and on glossy surfaces.

The contribution of our paper consists in formulating the algorithm of *bi-directional photon mapping* (BDPM) capable of handling various scenes with prevailing highly glossy materials without exhibiting the aforementioned artifacts. We do not attempt to address the former issue by increasing the number of photons. Such approaches are described for instance in [4, 5]. Instead, we deal with the latter by replacing the final gather heuristic by a more principled approach. The original photon mapping performs the radiance estimate from the photon map only at the end of the final gather rays and differ between "global" and "caustic" photon map while we use a combination of various path connection strategies corresponding to a photon map estimate performed at different vertices of the full camera path. Our approach is inspired by Veach's multiple importance sampling technique [1] for adaptive light path connection used in bi-directional path-tracing.

Figure 1 shows an example of two different strategies used for computing the light transport of length three and the result of their combination by BDPM. Image 1a) demonstrates the high variance of radiance estimate on glossy surfaces while the image 1b) shows how this variance exhibits itself as a grainy noise when one level of distribution ray-tracing is used to render diffuse surfaces. Image 1c) demonstrates the superior image quality produced by our bidirectional photon mapping.

The BDPM algorithm is essentially the combination of path-tracing algorithm which follows the light paths from camera and with photon mapping algorithm which trace light paths from light sources. The photon map query is performed in every vertex along the path from the camera. To avoid computing the same light transport multiple times weighting functions summing to unity are used with each photon-camera path pair. This is analogous to the approach taken in BDPT [1, 2].

In the following section we review the Photon Mapping algorithm. To be able to combine various strategies for computing the same light transport we need to treat each connection of a single photon with a path from the camera as a single path. The weighted contribution to the image pixel is computed along that path. To be able to do that, section 2.1 gives a formulation of photon mapping consistent with Veach's path integration framework [1, chapter 4.A]. This formalism allows us not to think about the photon mapping in terms of radiance estimate but rather in terms of individual paths which can be sampled from various strategies. Based on this fact we derive the formula describing the BDPM algorithm in section 3. Section 4 gives an overview of the algorithm and specifies some details about computing path weights. Finally, in section 5 we present our results.



a) PM, no final gather b) PM, final gather c) BDPM (our method)

Figure 1: *All images show rendering of the same scene and the same light transport of length exactly 3. The scene consists of diffuse walls and two highly glossy objects. The image to the left was rendered without final gather while the image in the middle uses final gather everywhere. Note how each strategy works well for different parts of illumination. The image to the right demonstrates combination of both strategies by our bidirectional photon mapping method.*

2 Background: Photon Mapping

The original photon mapping algorithm [3] consists of two passes. In the first pass, little quanta of energy called photons are emitted from the light sources and traced through the scene. When the photon hits a surface we store the information about the hit into a data structure called photon map. At the end of the first pass the information stored in the photon map approximates the overall global illumination of the scene.

In the second pass we cast camera rays into the scene through the image plane to compute the pixel values and whenever the ray hits the surface we can either compute the reflected radiance by means of distribution ray-tracing, i.e. estimating the integral over the hemisphere by casting secondary rays, or we can exploit the information in the photon map and estimate the outgoing radiance value. According to the formula

$$L(x,\omega_o) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x,\omega_o,\omega_p)\Phi_p(x,\omega_p) \qquad (1)$$

where we estimate the outgoing radiance from x in the direction ω_o, the radiance estimate procedure can be interpreted as expanding the sphere around the point x until we catch N photons within the disk of radius r [3]. Each photon coming from the direction ω_p found within the disk contributes by its energy Φ_p multiplied by the BRDF f_r evaluated right at point x. This interpolation step introduces bias into the image which exhibits itself as a low-frequency noise observed as blurriness in the image.

2.1 PM in Terms of Particle Tracing

We describe our bidirectional photon mapping algorithm within the framework introduced by Veach [1]. This allows us not to think in terms of an aggregate radiance estimate but rather in terms of contributions of individual paths sampled from various strategies. Under these conditions we are able to apply multiple importance sampling for weighting the contribution of various paths. In this section we relate the radiance estimate procedure described by Jensen to the particle tracing characterization described by Veach.

In [1, chapter 4.A] Veach describes particle tracing as a method which generates a set of N sample particle paths ρ_p with their corresponding weights denoted as α_p. From now on to avoid any confusion we will rather use a term photon instead of particle. Sample paths are constructed by following a random walk of a photon through the scene. Their corresponding weights are computed by multiplying the initial energy of a photon in each bounce by the appropriate BRDF value and cosine term and divided by the probability of sampling a new direction and probability of terminating the particle path.

In general each path ρ has its own importance $W_e(\rho)$ for the measurement which the photon is used for. Using Monte Carlo estimation general measurement can be expressed as a weighted sum over sample paths

$$I = E\left[\frac{1}{N}\sum_{p=1}^{N} W_e(\rho_p)\alpha_p\right]. \qquad (2)$$

We implement this measurement as a radiance function value estimation. To estimate $L(x,\omega_o)$ we define $W_e(x,\omega_p) := f_r(x,\omega_o,\omega_p)$. ω_p denotes the direction of the incident photon in point x:

$$L(x,\omega_o) = E\left[\frac{1}{N}\sum_{p=1}^{N} f_r(x,\omega_o,\omega_p)\alpha_p\right]. \qquad (3)$$

In this formulation, photons which end their path out of the point x will yield no contribution.

Last step towards the photon mapping within Veach's formulation is using the biased estimator in form of den-

sity estimation method [6] instead of the unbiased one. Introducing the kernel

$$\kappa(d) = \begin{cases} 0 & \text{if } d > r \\ \frac{1}{\pi r^2} & \text{if } d \leq r, \end{cases} \tag{4}$$

where r is the radius, the formula for the radiance estimation now reads

$$L(x,\omega_o) \approx E\left[\sum_{p=1}^{N} \kappa(\|x-x_p\|) f_r(x,\omega_o,\omega_p)\alpha_p\right]. \tag{5}$$

The point x_p represents the last vertex on the photon path ρ_p (i.e. the photon position) with corresponding weight α_p. This equation suggests that only those photon paths which end within the support of kernel κ will contribute to the radiance estimate.

Of course, we ended basically with the same formula as we have already stated in (1). The purpose of this section was to put the radiance estimate into the context of Veach's light transport framework that servers as basis for our BDPM formulation.

3 BDPM formulation

Veach in his dissertation presented the path integral formulation which is the measurement equation describing the whole light transport in the form of one non-recursive equation where the computation is concentrated around a geometric primitive - the path [1]. In this framework he derived Bi-Directional Path Tracing originally introduced by Lafortune [2], an unbiased algorithm in which he combines paths from light sources and paths from the camera. After connecting one path from the camera to one path from the light source, the contribution of the resulting path is weighted so that no light transport is taken into account more than once. Each strategy, defined by the length of the light and camera paths, is better at computing some parts of the whole light transport while being worse at computing the other. Nevertheless, each of them converges to the correct result. Their combination just exploits their respective advantages and makes the overall light transport computation faster and more robust.

In Photon Mapping we can interpret the radiance estimate procedure as connection of one path from camera to N paths from the light source as opposed to BDPT where paths are connected in a one-to-one manner. To be able to apply Multiple Importance Sampling we need to separate these N paths which share the same camera prefix and then weight each path alone.

In the following section we show the derivation of the formula which describes the BDPM algorithm itself. Overview of the algorithm is given in section 4.

3.1 Derivation

We aim to combine the path-tracing algorithm with the photon mapping. To describe path-tracing we use a standard Monte Carlo quadrature which describes the measurement of discrete pixel value I_j of pixel j

$$I_j = E\left[\frac{1}{M}\sum_{l=1}^{M} T_l L_e(x,\omega_o)\right] \tag{6}$$

where T_l is a throughput of the whole camera path ρ_l^* and is computed by multiplying the initial importance value of that path in every bounce by appropriate BRDF and cosine term and divided by the probability of sampling the new direction and by the probability of terminating the path. Point x is the last vertex on the camera path ρ_l^* and ω_o is the direction pointing towards the direction from which the last segment was sampled. L_e is the source radiance function. Note the obvious duality to the particle tracing described in previous section.

Let $M = 1$ so that we have the following one-sample estimator

$$I_j = E\left[T L_e(x,\omega_o)\right] \tag{7}$$

and let us consider the scenario where we have only photons which bounced along the paths ρ_p of the same length s segments and its corresponding weights are α_p^s. From the camera point of view we will take into account only one path from camera ρ^* of t segments with its throughput T^t.

Under these conditions we can perform connection of light sub-paths by replacing L_e in the equation (7) by equation (5) so that we compute only the pixel value I_j^n due to the light transport of length $n = s+t$

$$I_j^n \approx E\left[T^t \sum_{p=1}^{N} \kappa(\|x-x_p\|) f_r(x,\omega_o,\omega_p)\alpha_p^s\right]. \tag{8}$$

We cannot write the equal sign because using the radiance estimate introduces bias into the pixel value estimation. The equation describes the process of tracing the path of fixed length from the camera and then querying the photon map in point x. In the photon map query only those photons are taken into account which traveled along the path of fixed length. End points x and x_p as well as the directions ω_o and ω_p depend on the sampled sub-paths and for simplicity will be omitted in the following formulas. By simple multiplication we get the biased estimator for I_j^n

$$I_j^n \approx E\left[\sum_{p=1}^{N} \kappa T^t f_r \alpha_p^s\right]. \tag{9}$$

In this formula we put the term T^t inside the sum to emphasize that the formula treats all N paths in the estimator as N separate light paths sharing the same camera subpath.

Next we will introduce the following notation

$$X_{s,t} := \rho_p^s \rho^{*t} \tag{10}$$

to depict the path consisting of s segments from light source and t segments from camera.

Now we can apply Veach's multiple importance sampling technique (MIS) to combine more estimators in form of (9) so that we account for contributions of paths with various length of sub-paths from camera and from light source. Our multi-sample estimator looks as follows:

$$I_j^n \approx E\left[\sum_{i=1}^{n}\sum_{p=1}^{N} w(X_{n-i,i})\kappa T^i f_r \alpha_p^{n-i}\right], \qquad (11)$$

where $w(X_{n-i,i})$ is MIS weight function for path of length n and index i runs through n various strategies where i means the number of segments on the path from camera before it is connected with paths from light.

Knowing how to weight the path contributions we can rearrange the quadrature so that we can use the N-nearest neighbour query into the photon map after tracing the path from camera:

$$I_j^n \approx E\left[\sum_{i=1}^{n} T^i\left(\sum_{p=1}^{N} \kappa w(X_{n-i,i}) f_r \alpha_p^{n-i}\right)\right]. \qquad (12)$$

The expression in the round brackets is the radiance estimate restricted to fixed photon path length and enriched by MIS weight function which takes into account even the sub-path traced from camera.

It was necessary to fix the sub-paths length to show how to weight the individual path samples. If we consider the fact that

$$I_j = \sum_{i=1}^{\infty} I_j^i \qquad (13)$$

and that the sub-path length both from camera and from light are random variables which in the implementation are determined by using Russian Roulette then we can remove the restriction of equal photon path length in the radiance estimate and simplify the formula to

$$I_j \approx E\left[T\left(\sum_{p=1}^{N} \kappa w(X_{s(p),t}) f_r \alpha_p\right)\right]. \qquad (14)$$

Using Russian Roulette for controlling sub-path lengths gives non-zero probability of sampling the light path of any finite length.

In the next section we give an overview of the algorithm which is essentially the evaluation of the formula (14) and we describe how to deal with evaluating the MIS weight function w.

4 Overview

Our algorithm is very similar to the original photon mapping (PM). It consists of two passes. In the first pass we collect photon histories in one photon map (i.e. unlike in PM, there is no special "caustic" photon map and "global" photon map) with additional information about probabilities of sampling photon paths. In the second pass the distribution ray-tracing and the final gather heuristic, as it was described by Jensen [3], is replaced by standard path-tracing. The special radiance estimate procedure involving weighting each photon contribution is performed gradually at every vertex of the path traced from camera.

The described algorithm is based on the formula (14) and in this section we especially focus on evaluating the weight function w. In next two sub-sections we describe how to compute probabilities needed for evaluating w. Evaluation itself is then described in sub-section 4.3.

4.1 First Pass

We start shooting and tracing the photons as in original PM including usage of Russian Roulette. The only difference is that we need to store probabilities of generated photon paths which are used during the second pass for computing the MIS weights.

Let $y_0,\ldots,y_s$ denote vertices of a photon path ρ and let y_0 be a point on a light source. Then we define probability p_k^L of sampling the path $y_0,\ldots,y_k$ for $0<k\leq s$ as

$$\begin{aligned} p_1^L &= p_A(y_0)\cdot p_D(y_0\to y_1)\\ p_i^L &= p_{i-1}^L\cdot p(y_{i-1},y_{i-1}\to y_i); \qquad \forall i \quad 2<i\leq k \end{aligned}$$

where $p_A(z)$ is the probability of sampling a point z on the light source, $p_D(\omega)$ describes the directional properties of the light source and finally $p(z,\omega)$ is the probability of sampling the direction ω from the point z.

We store the probability p_k^L with each photon hit record in the point y_k. Another information that we need to retain for the second pass is the track of photon bounces within single photon path. For example for a given point y_i we need to be able to backtrack the probabilities $p_{i-1}^L\ldots p_1^L$ to evaluate the path weight during the second pass.

4.2 Second Pass

The backbone of the second pass is the path-tracing algorithm. We incrementally construct the path ρ^* from camera using Russian Roulette for controlling the path length. In every vertex we perform a special radiance estimate as described by formula (14) in the round brackets. To be able to evaluate the weight function w we need to determine the probability of sampling the current camera path.

Let us suppose that we have already constructed the path $x_0,\ldots,x_k$ and updated the throughput T of that path. Then we need to compute the probability p_k^E of sampling the path for $0<k\leq t$ as follows:

$$\begin{aligned} p_1^E &= p_A(x_0)\cdot p_D(x_0\to x_1)\\ p_i^E &= p_{i-1}^E\cdot p(x_{i-1},x_{i-1}\to x_i); \qquad \forall i \quad 2<i\leq k. \end{aligned}$$

Similarly as in the case of light source p_A and p_D depends on the type of camera. Usually when using pinhole camera we set $p_1^E=1$. The $p(z,\omega)$ is the probability of sampling the direction ω from the point z.

```
proc secondPass(j) ≡
  I_j := 0;
  while (ray := generateRay(j)) do                Ray through the pixel j
      T := W_e^j(ray);                            source importance of the ray
      p_1^E := 1;                                 1 for case of pinhole camera
      k := 0;
      while ¬absorption(x_k) do                   apply RR
          k := k + 1;                             prolong the path
          x_k := intersect(ray);
          p^E = array(p_1^E, ..., p_k^E);
          comment: compute pixel contribution
          comment: R is weighted radiance estimate procedure
          I_j := I_j + T · R(radius, k, p^E);
          comment: Generate new ray with probability "pdf"
          ray := sampleBRDF(x_k, var pdf);
          pdf := pdf * RRpdf;                     includeRRprobability
          comment: update throughput and compute path probability
          T = T * BRDF(x_k) * cosθ / pdf;
          p_{k+1}^E = p_k^E * pdf;
      od
  od
end
```

Figure 2: *Pseudo-code for the second pass of our bi-directional photon mapping algorithm. Procedure computes the value I_j of pixel j.*

After updating the throughput T and determining the probability p_k^E we can perform the special weighted radiance estimate according to formula (14) and add the contribution to overall pixel value I_j.

If the camera path is not terminated due to application of Russian Roulette we extend the path about new vertex x_{k+1} by tracing the ray from the point x_k and we apply the same procedure as for the vertex point x_k. We prolong the path in the same manner until the absorbtion of path occurs. The pseudo-code for the second pass is shown in Figure 2.

4.3 MIS Weighted Radiance Estimate

The procedure for estimating the radiance from the photon map is basically the same as in the classical photon mapping algorithm. The biggest difference is that each photon contribution is multiplied by MIS weight dependent both on the given photon sub-path and on the current camera sub-path. In the rest of this section we describe how the MIS weight is computed.

During the algorithm we take path samples from various strategies. The probability of sampling some path x from the strategy where we connected camera sub-path with t segments to photon sub-path with s segments is

$$P_{s,t}(x) = p_t^E p_s^L, \tag{15}$$

where p_t^E is probability of sampling first t segments of path x from camera and p_s^L is probability of sampling the rest s segments from a light source.

For sampling the paths of length n we use $n-1$ strategies in our implementation since for now we neglect two

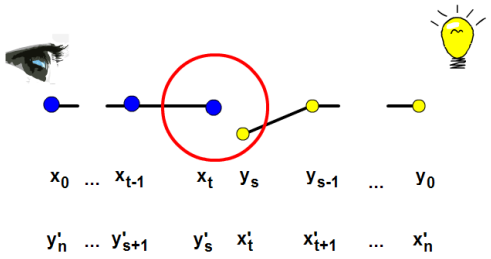


Figure 3: *This figure clarifies the extended notation of a sampled path used for defining the sub-path probabilities. The red circle depicts the kernel centered in the point x_t.*

corner cases where there is no segment on a camera sub-path or no segment on a light sub-path (see section 6). Using balance heuristic for MIS [1] every strategy has its corresponding weight function

$$w_{s,t}(x) = \frac{P_{s,t}(x)}{\sum_{i=1}^{n} P_{n-i,i}(x)}. \tag{16}$$

For the path $X_{s,t}$ we simplify the notation to

$$w(X_{s,t}) = w_{s,t}(X_{s,t}) \tag{17}$$

so that we are consistent with section 3.

In previous section we described how to compute sub-path probabilities $p_1^E, \ldots, p_t^E$ and $p_1^L, \ldots, p_s^L$. To be able to evaluate MIS function w obviously we need to evaluate $p_{t+1}^E, \ldots, p_{n-1}^E$ and $p_{s+1}^L, \ldots, p_{n-1}^L$.

Let assume that we have sampled one path

$$X_{s,t} = x_0, \ldots, x_t, y_s, \ldots, y_0$$

as it was denoted in the preceding text. The camera sub-path starts at x_0 and we currently perform the radiance estimate in its end point x_t. Photon sub-path starts at y_0 and ends at y_s where it was found within the support of the kernel κ. We introduce the redundant notation

$$x_0 = y_n', \ldots, x_{t-1} = y_{s+1}', x_t = y_s',$$
$$y_s = x_t', y_{s-1} = x_{t+1}' \ldots, y_0 = x_n'$$

so it is possible to easily express how to compute the rest of sub-path probabilities. This notation is clarified in Figure 3.

Then we take special care about probabilities

$$p_{s+1}^L = p_s^L \cdot p(x_t, y_{s-1} \to y_s, y_s' \to y_{s+1}')$$
$$p_{t+1}^E = p_t^E \cdot p(x_t, x_{t-1} \to x_t, x_t' \to x_{t+1}')$$

where we compute the probability in point x_t in both cases. The reason is that the radiance estimate is performed in point x_t and thus the BRDF f_r is evaluated at x_t for the given photon. Note that in the limit case when there was infinite number of photons the points x_t and x_s would merge into one point.

Other sub-path probabilities

$$p^L_{s+i} = p^L_{s+i-1} \cdot p(y'_{s+i-1}, y'_{s+i-2} \rightarrow y'_{s+i-1} \rightarrow y'_{s+i}) \quad \text{for } i \in \langle 2, t \rangle$$

$$p^E_{t+i} = p^E_{t+i-1} \cdot p(x'_{t+i-1}, x'_{t+i-2} \rightarrow x'_{t+i-1} \rightarrow x'_{t+i}) \quad \text{for } i \in \langle 2, s \rangle$$

can be computed all in the same manner. They can be even pre-computed and stored in a cache during tracing the camera sub-path and tracing photons respectively because the outgoing and incoming directions are known at that time.

To evaluate MIS weight we need to compute all $\sum_{i=1}^{n} P_{n-i,i}(X_{s,t})$. It is easy to evaluate $P_{s,t}(X_{s,t})$ for the current path because from given photon history and camera path-tracing we immediately get p^L_s and p^E_t respectively. The rest sub-path probabilities can be evaluated starting from these ones.

In this stage we know everything to evaluate the weight for a given single path according to formulas (16) and (15) and thus to evaluate the weighted radiance estimate. With this knowledge we can compute the contribution to the pixel value I_j according to formula (14) as it is described in pseudo code in figure 2.

5 Results

We provide renderings of a challenging scene in terms of the number of glossy surfaces and light settings. All test images were rendered on a 1.6GHz Intel Core i7 Q720 using 4 physical cores but 8 logical cores. We ran one thread per logical core.

Figure 5 shows a model of a kitchen where almost every material is glossy. A number of spot lights are used as the only source of the light in the scene. They are all placed on the opposite site of the kitchen behind the camera and all light reaching the shot went through at least one indirect bounce, so there is no direct lighting in the particular shot. To make the conditions even more difficult there are few bottles made of frosted glass placed on the counter. In all images we restricted the path length to a maximum of 7 segments.

These renderings demonstrate the robustness of the BDPM which is its main advantage over the state-of-the-art. PM with final gather will always sample paths only from the one and only strategy and thus it can always be given some scene (as in our case) on which it will perform very poorly.

The image 5a) shows result of path-tracing using 131 072 samples per pixel. Even though it is still noisy it can serve as the reference image to judge which reflections in the images 5b) and 5c) are correct and which are artifacts.

The image 5b) was rendered by standard photon mapping with final gather heuristic. We used one level of distribution ray-tracing (final gather) casting 128 secondary rays and 32 samples per pixel. This means performing $128 \times 32 = 4096$ radiance estimates per pixel.

To get comparable results we used 512 samples per pixel for rendering the image 5c) which was rendered by BDPM so we would perform $512 \times 7 = 3584$ radiance estimates and thus took approximately the same number of measurements in both 5a) and 5b) and run the algorithms for the same amount of time. In both cases we used 2 million photons, nearest-neighbour density estimate in the photon map query with the conical kernel [3].

We can see that in the image 5c) the severe noise on the cupboard vanished completely. Important improvement is also observable on the tea pot. Unlike the image 5c) on the image 5b) in the counter there is not visible the reflection of the main reflection visible on the tea pot. Given the circumstances this will not get better in the classical photon mapping algorithm even if we used more samples in the distribution ray-tracing. The reason is almost specular surface of the tea pot. We would have to dramatically increase the number of photons in the map to get better results in 5b). Another noticeable issue are some completely missing reflections.

In figure 6 we present renderings of a Cornell box-like scene. Diffuse materials are used for walls and highly glossy materials for the blocks. The images were rendered with paths of length exactly three (i.e. $n = 3$). Image 6a) represents the strategy where $t = 1$ and $s = 2$ while the image 6b) shows the result of using strategy where $t = 2$ and $s = 1$. Images 6c) and 6d) were rendered by bidirectional photon mapping where both of the latter strategies were combined together. We used 20 million photons to render the combined image and 512 samples per pixel. In image 6c) we used fixed radius for every photon map query while in the image 6d) we used the nearest-neighbour estimation method and conical kernel. This demonstrates that using varying radius through the scene does not yield objectionably worse results.

6 Conclusions and Future Work

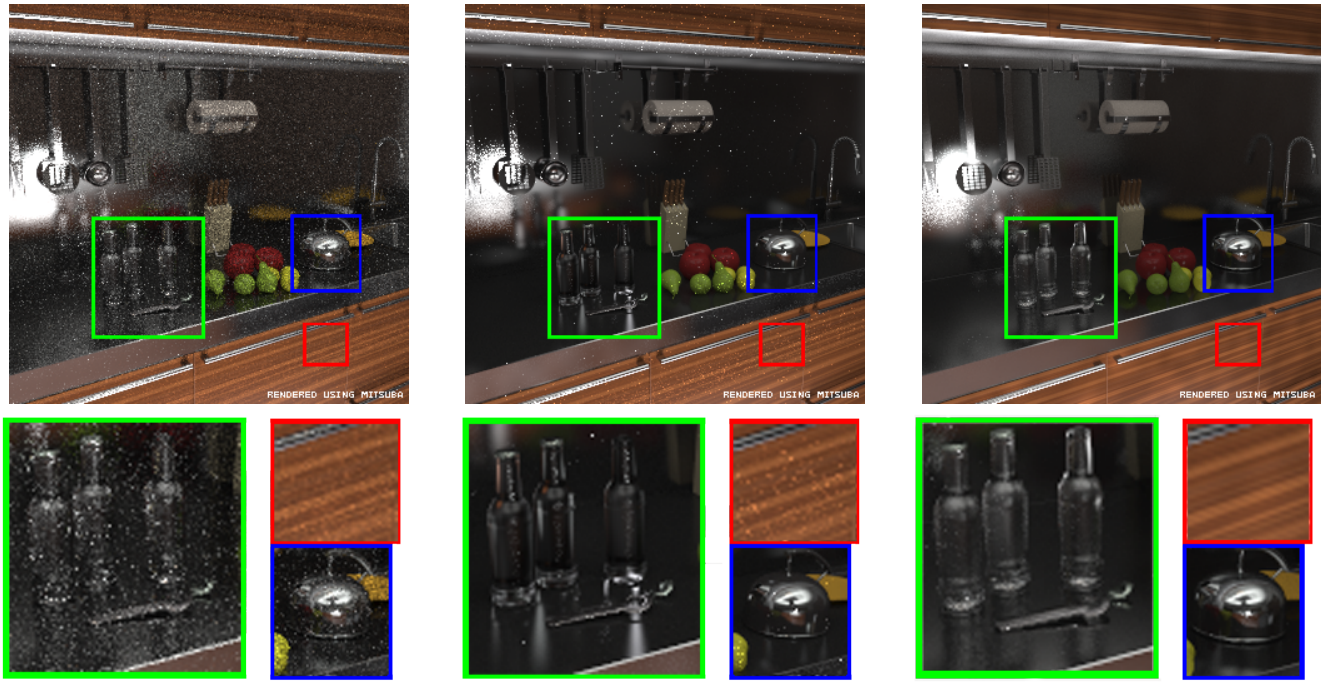
We have presented a robust bidirectional photon mapping algorithm for computing global illumination. The algorithm is capable of rendering scenes with many glossy materials where the original photon mapping algorithm produces objectionable artifacts. The algorithm takes advantage of combining various strategies for computing the same light transport.

In our implementation we ignore strategies where the path is generated entirely from the light source or from the camera. Taking these paths into account will be addressed in future work.

One reason for artifacts in glossy scenes is the high variance of radiance estimate on glossy surfaces. This can be improved by shooting an enormous number of photons. Nevertheless, original photon mapping is limited by physical memory boundaries so we cannot use sufficient num-

Algorithm	Samples per Pixel	Final Gather Samples	Resolution	Rendering Time
PT	131 072	-	320x320	1.4 day
PM	32	128	320x320	4 hours
BDPM	512	-	320x320	4.1 hours

Figure 4: Information about renderings in figure 5.



a) Path tracing (PT) b) Photon mapping (PM) c) Our method (BDPM)

Figure 5: *There is a comparison of rendering the same scene by PT, PM and BDPM. We let PM and BDPM run for the same time. Both PM, BDPM use 2 million photons, nearest-neighbour density estimate and conical kernel. Noisy reference image was rendered by PT with 131 072 samples per pixel.*

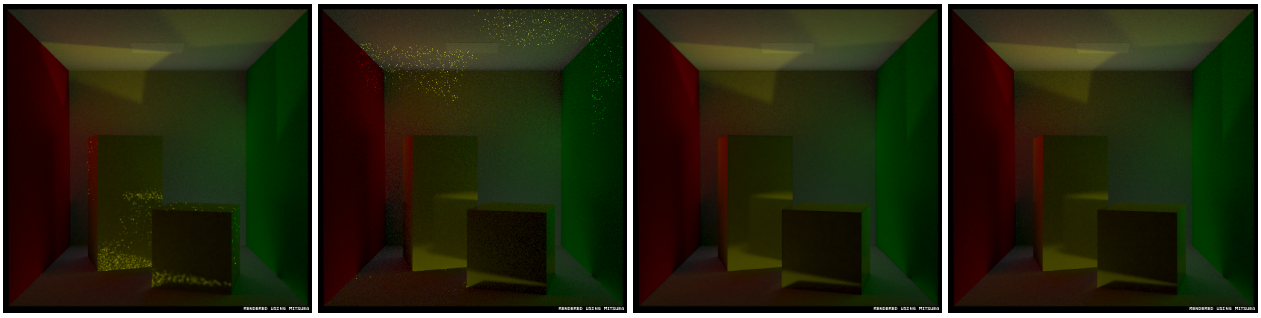


Figure 6: *All images show rendering of the same scene and the same light transport of length 3. There are diffuse walls and two highly glossy objects. The image a) was rendered without final gather while the image b) uses final gather everywhere. Note how each strategy works well for different parts of the illumination. Images c) and d) demonstrate combination of both latter strategies by bidirectional photon mapping method. In c) we used fixed radius in density estimate through the whole scene while in d) we allowed varying radius and used conical kernel.*

ber of photons. This boundaries are addressed by progressive photon mapping approaches [4, 5]. In future work we would like to examine the possibilities of bidirectional photon mapping within these progressive frameworks.

Acknowledgements

Many thanks to Wenzel Jakob for providing his Mitsuba Renderer. I am no less grateful to Jaroslav Křivánek for his supervision, numerous advices, ideas and text corrections.

References

[1] Eric Veach, *Robust Monte Carlo Methods For Light Transport Simulation*. Ph.D. Dissertation, Standford University, 1997.

[2] Eric P. Lafortune, Yves D. Willems, *Bi-Directional Path Tracing*. Proceedings of Compugraphics '93, Alvor, Portugal (December 1993), pp. 145-153.

[3] Henrik Wann Jensen, *Realistic Image Synthesis Using Photon Mapping*. AK Peters, 2001.

[4] Toshiya Hachisuka, Shinji Ogaki, Henrik Wann Jensen, *Progressive Photon Mapping*. ACM Trans. Graph. 27,5, 2008.

[5] Toshyia Hachisuka, Henrik Wann Jensen, *Stochastic Progressive Photon Maping*. ACM SIGGRAPH Asia, 2009.

[6] Shirley P., Wade B., Hubbard P. M., Zareski D., Walter B., Greenberg D. P. *Global illumination via density-estimation*. Eurographics Rendering Workshop 1995 Proceedings, pp. 219–230.

Rendering

Workflow Optimization for a Graphic Artist working on large Texture Sets using Virtual Texturing

Michael Birsak*
Supervised by: Michael Wimmer†

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

Abstract

In this paper we present an approach to optimize the workflow for a graphic artist currently working on high resolution photographs of architectural and archaeological monuments. These photographs are used as texture maps to color the meshes, which are calculated from laser-scanned point clouds corresponding to exactly those monuments. Because a particular region belonging to a monument is covered from several photographs with different colorization, further manual processing of the photos is required. Therefore we developed an application, which generates masks to emphasize regions of the photographs that are used in the final model, to ease the work of the graphic artist. For fast rendering of the model, we took Virtual Texturing into account, and developed an application for fast generation of the Virtual Texture Atlas and the Tile Store. A fast and efficient update of the Tile Store, if a photograph is edited after the final generation of the Virtual Texture Atlas and the Tile Store, the application also provides. The used algorithms are stated as well as the speed-up compared to an existing implementation.

Keywords: Mask Generation, Virtual Texturing, Texture Atlas, Fast Tile Update

1 Introduction

At the Vienna University of Technology, the aim of the Terapoints-Project[1] is the preservation of important architectural and archaeological items. This preservation is done by laser-scanning these items, that yields huge point clouds. Further, photographs with registered cameras are taken, to make a colorization of the digitized model possible. To allow rendering of the model inside a standard tool like Meshlab [2], the point cloud is transformed into a triangle mesh using the algorithm from Abdelhafiz [1]. This has the advantage not to be constrained to applications implementing algorithms like Instant Points from Wimmer and Scheiblauer [8]. Further, a mesh allows a continuous mapping of the photographs onto the model. The different lighting situations depending on the different scanning positions lead to colorization differences between the photographs. For that reason, processing of these photographs is necessary. At the moment, all processing steps are done by a graphic artist, which means that every photograph contained in the final model has to be edited in a graphics editing application like Adobe Photoshop[2]. Without further processing, there would be visible artifacts in the model, where two regions, belonging to two different photographs, adjoin each other. In Figure 1 you can see such artifacts.

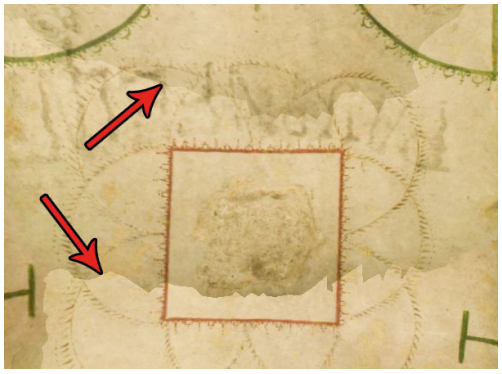


Figure 1: Visible artifacts in the Domitilla model without further processing of the photographs. The arrows show the regions, where different photographs adjoin each other.

The rendering of the whole model is important for the graphic artist to see all the photographs and the possible artifacts in action. Currently, the rendering is happening in MeshLab. The current workflow of the graphic artist is shown in Figure 3.

Because of the high resolution and the large quantity of photographs in a single 3D model, there is a need to accelerate the rendering of the whole model. Without this acceleration, there is an unnecessary amount of traffic between the CPU and the GPU during the rendering process, because not all photographs fit into the memory of the graph-

*michael.birsak@gmx.at
†wimmer@cg.tuwien.ac.at
[1] http://www.cg.tuwien.ac.at/research/projects/TERAPOINTS/
[2] http://www.adobe.com/de/products/photoshop/compare/

Figure 2: Part of the Domitilla Catacombs visualized using the unprocessed photographs. Especially on the floor the aforementioned artifacts are visible.

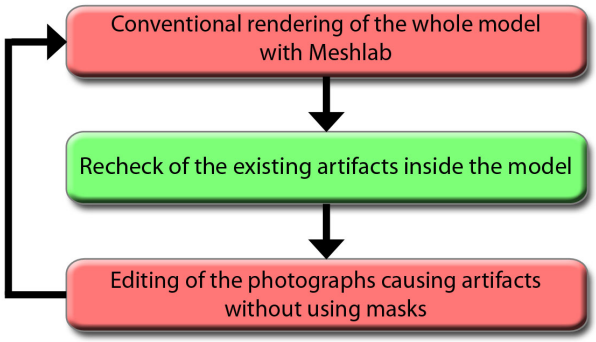


Figure 3: Current workflow of the graphic artist. The first and third step in the cycle are the ones, we want to optimize.

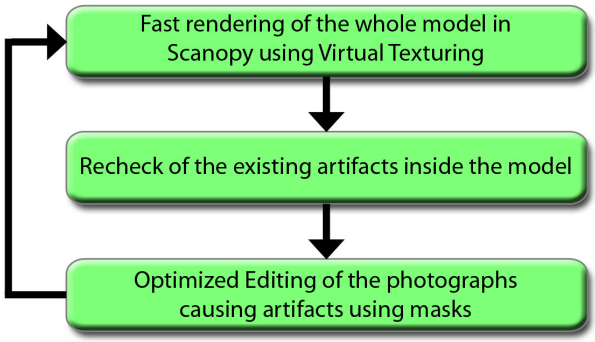


Figure 4: Optimized workflow of the graphic artist. The rendering is accelerated using Virtual Texturing, the editing is eased by the generated masks.

ics card. The results are low frame rates and therefore smaller productivity of the graphic artist. Due to this we decided to use *Virtual Texturing*, where only those parts of all photographs reside in the memory of the graphics card, which are currently visible.

As application to visualize the virtual textured model, we chose Scanopy because it implements LibVT[3] (developed by J. Mayer, the author of [5]), a library which provides Virtual Texturing functionality. Scanopy is developed at the Vienna University of Technology and at the Imagination[4] in Vienna. We implemented additional functionality into Scanopy, to directly call our application via keystroke to update the *Virtual Texture Atlas* (in the following just referred to as *Atlas*) and *Tile Store* when a photograph has changed. So, in future the graphic artist will use Scanopy for faster rendering of the model. The optimized workflow of the graphic artist, introducing Virtual Texturing for faster rendering and masks for the photographs for easier editing, is shown in Figure 4. In Figure 2 you can see a part of the Domitilla Catacombs, on which the graphic artist is currently working on.

[3] http://sourceforge.net/projects/libvt/

[4] http://www.imagination.at/

Contribution. In this paper, we present two applications to optimize the current workflow of the graphic artist, who edits the photographs of the monuments. The first program generates masks for the photographs belonging to a model. These masks consist of black and white areas, where white areas correspond to areas in the photographs, which are visible in the model, whereas black areas correspond to invisible areas in the photographs. The generated masks can be used in every graphics editing program, to avoid editing areas of the image which are invisible in the model.

The second program is used for fast generation of the Atlas and the Tile Store. Although there already exists an implementation for this task, we found it too slow to meet the requirements of the graphic artist concerning time consumption. Our program is also used for updating the Atlas and the Tile Store, when a photograph is changed after the Atlas and the Tile Store are generated.

In Section 2 we give some background information about Virtual Texturing and the existing implementations for the generation of the Atlas and Tile Store. In Section 3 and 4 we will give detailed information about the development of the applications we have implemented. Finally, in Sections 5 and 6 we present the results concerning our applications as well as possible additions which can be done in future.

2 Related Work

Virtual Texturing, as it is described detailed by J. Mayer in [5], is a sophisticated technique to overcome the memory limitations of the graphics card, when rendering objects or scenes with a large quantity of textures. Rendering such scenes might work without Virtual Texturing as well, but the frame rate would probably suffer extremely under the high traffic rate between the CPU and GPU, resulting from continuous streaming of texture data. In contrast to Virtual Texturing, every needed texture would be loaded as a whole, regardless of the visible texture area. With Virtual Texturing, only those parts of the texture are streamed to the graphics card, which are actually visible. The smallest part, that can be streamed to the GPU, is called a *tile*. A tile is a small texture with a resolution of 64^2 up to 512^2 pixels, so its final side length must be of the form 2^n pixels for $n \in \{ 6, 7, 8, 9 \}$. If just one pixel of a texture is needed to render the scene, at least the whole tile containing this pixel must be streamed.

The first step for Virtual Texturing is to produce one big texture, consisting of all the single textures in the scene. This texture is the Atlas. In Figure 5 an example of such an Atlas is shown. The Atlas must fulfill some requirements regarding its size (refer to [5] for details). Because the Atlas often has side lengths of 32k pixels and more, it would be very unhandy to store it in a single file. Therefore, it is stored in 4, 16 or more equally sized files.

Note, that it is important to adapt all texture coordinates

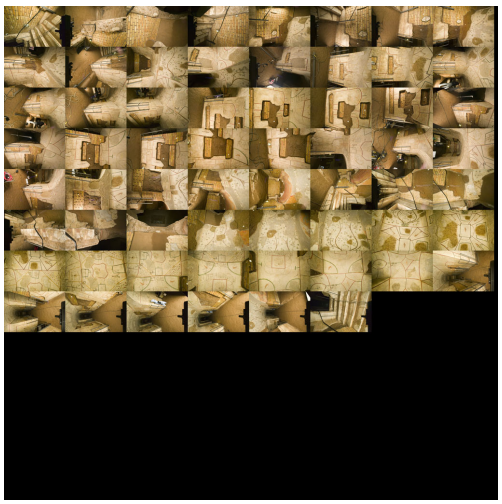


Figure 5: $32k^2$ Atlas consisting of 62 4064x2704 photographs of the Domitilla model.

of the models in the scene to refer to the layout of the Atlas. The Atlas is the base of the Tile Store. The Tile Store can be viewed as a mipmapped Atlas. In contrast to classic Mip Mapping [7], where the side lengths of the original textures are halved until only one pixel resides, the smallest part, representing the highest level of the Tile Store, is one single tile. If we have, for example, an Atlas with a resolution of $32k^2$, and a tile resolution of 128^2, there would be 65536 tiles at Level 0 of the Tile Store. At Level 1, there would be 16384 tiles and so on. At the highest Level with number 8, there would just be one tile representing the whole Atlas. In Figure 6 you can see exactly this scenario applied to the Atlas of Figure 5.

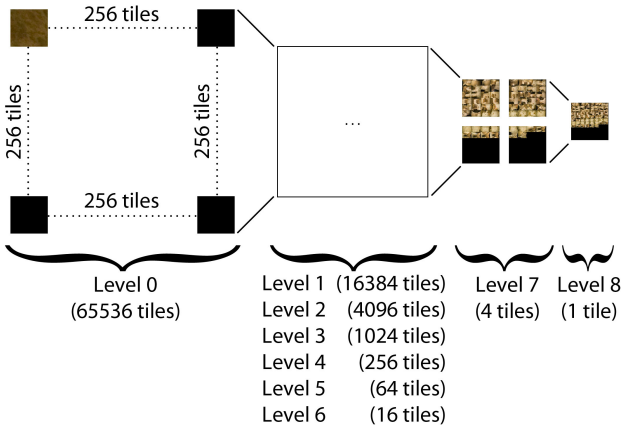


Figure 6: Tile Store generated with the Atlas shown in Figure 5 as its base. The tiles have a resolution of 128^2.

J. Mayer, the author of [5], has already implemented scripts to generate the Atlas and the Tile Store. Therefore he chose Python as scripting language. His script to generate an Atlas is based on ImageMagick[5], a command line

[5] http://www.imagemagick.org

based graphics editing program. The script to generate the Tile Store uses the Python Imaging Library. We will show, that our application, fully implemented in C++, accelerates the task significantly.

3 Mask Generation

The first step to ease the work of the graphic artist, was the development of an application to generate masks for every photograph contained in the model to prevent the editing of invisible image areas. The masks have exactly the same resolution as the photographs, so that every white pixel of a concrete mask corresponds to a visible pixel, and every black pixel corresponds to an invisible pixel in the underlying photograph. To generate a mask for a texture, the only information needed from the model are the face indices into the texture coordinates list, the texture coordinates themselves as well as the material information, to know which primitives belong to which photograph. The face indices are needed, to know which of the texture coordinates belong together to form a primitive, e.g. a triangle. The texture coordinates are values $\in [0, 1]^2$. Therefore it is important to choose a virtual camera (via the projection matrix), whose image plane corresponds to the whole area of possible texture coordinates. The camera, which fulfills exactly these requirements, is an orthographic camera placed in the world coordinate system at position $\mathbf{p} = (0.5, 0.5, 0.0)$. The width and height of the view frustum must both be 1. Since the simplest way to render 2D content is to use the XY-plane, the values for the near and far clipping plane can be set to an arbitrary negative and positive value respectively. In Figure 7 the view frustum of this virtual camera is shown.

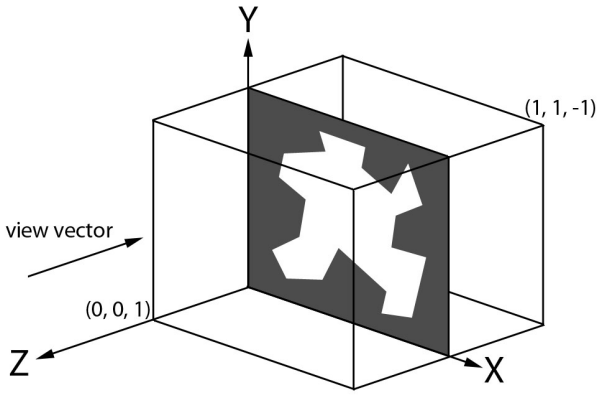


Figure 7: View frustum of the orthographic camera used to render the masks. All primitives are rendered into the XY-plane.

The masks are generated on the GPU using OpenGL as the graphics API. Therefore a buffer with the same dimensions as the current photograph is created. The background color is set to black, which corresponds to invisible image areas. After that, all primitives of the model are rendered in white color into the buffer. The texture coordinates of the primitives are used, as if they were vertex positions. Because texture coordinates are $\in [0, 1]^2$, the z-value is set to 0 to use the XY-plane as the plane to render into.

Our application to generate masks is fully implemented in C++, currently only available for Microsoft Windows. To make it small and simple, we omitted a graphical user interface. Per default, the mask generation application copies a shortcut to itself into the Windows SendTo directory during the installation. Due to this, it can simply be started by a right click onto the model file followed by a "Send To" to our application. After that, all masks will be generated in a sub directory called masks. Of course, all textures belonging to the model must be at the position referenced in the model file.

4 Virtual Texturing

The second step to optimize the workflow of the graphic artist was the introduction of Virtual Texturing for faster rendering of the models. Due to this, we had to improve the already existing scripts, developed by J. Mayer, the Author of [5], to generate the Atlas and the Tile Store. The existing scripts are implemented in Python. Because these scripts need a very long time, to produce the Atlas and the Tile Store (see Section 5 for concrete values), we decided to implement a new application, which should do the same task in much less time. In contrast to the existing scripts, which first generate an own Tile Store for every part of the Atlas, and then merge these to come to the final Tile Store, we decided to implement it in that way, that the final Tile Store is generated out of all the parts of the Atlas in a single run. Our application is also used for the update of the Atlas and the Tile Store, when the graphic artist has changed one or more of the photographs. Because this is the most time-critical function of our application, since it is used after every editing step of a photograph, we will explain it in detail in the following subsection.

4.1 Atlas- and Tile Store-Update

Because the generation of the Atlas and the Tile Store is, also when done with our fast application, a relatively time consuming task, we decided not to regenerate the Atlas and the Tile Store for an update, but to reproduce just those parts, which are concerned with the changed photographs.

To make an update of the Atlas and the Tile Store possible, it is important to know which photographs inside the Atlas have changed and where every photograph is positioned in the Atlas. To accomplish this, a text file is produced during the generation of the Atlas. This text file contains the time of last change, which can be queried from the operating system, of all produced Atlas files. If a photograph is changed and the update routine is run, the time stamp in the text file and the current time stamp would dif-

fer, indicating that all parts of the Atlas and the Tile Store belonging to this concrete photograph must be updated. The exact position of the photograph which has changed is so important to know, because otherwise it would not be possible (at least not without much effort) to find the areas in the Atlas and Tile Store the photograph belongs to.

When the changed areas of the Atlas have been found, it is easy to calculate the concrete coordinates of the parts of the Atlas, that are concerned. During the update all levels of the Tile Store have to be considered. Because every level corresponds to a mipmap level of the Atlas, the Atlas must be scaled down to the half after one particular level has been updated. As already mentioned in Section 2, the Atlas is of course not stored in one single file, but in 4, 16 or more parts. Therefore, there may be parts of the Atlas, which are not concerned by the update process. After the update of one particular level of the Tile Store, the Atlas must be scaled down to deliver the necessary information for the next level of the Tile Store. This would be redundant work for every update, if unchanged parts would be scaled every time. Due to this, our application not only stores the Atlas itself, but also every mipmap level needed. Certainly, this scaled parts of the Atlas must also be updated, if they consist areas of a photograph that was changed.

Tile Cache and VRAM. When updating the Atlas and the Tile Store, it is also important to update the dedicated memory region on the graphics card (*VRAM*), which is used to store the currently used tiles. Further, it is necessary to update the so called *Tile Cache*, which is the dedicated region inside the main memory, to hold a finite number of tiles to prevent another time consuming streaming from hard disk. Another streaming of a particular tile can be necessary when it has been overwritten inside the VRAM by another tile because of a certain time period without usage. A problem that arises, if such an update is not executed, is the simultaneous usage of the old and the new version of tiles corresponding to a particular photograph. This happens, because currently just those tiles are streamed from hard disk, cached, and then streamed to the graphics card, that are needed to render the next frame, but are not already stored inside the Tile Cache. When a particular tile is needed again, but is already stored in the Tile Cache, this version is streamed to VRAM, no matter if it has changed on hard disk.

In Figure 8 you can see a part of the Domitilla model rendered with Virtual Texturing inside Scanopy, showing visual artifacts when the Atlas and the Tile Store change, but an update of the Tile Cache and VRAM is not executed. To emphasize the artifacts, the old version of the photograph was patterned before the creation of the Atlas.

LibVT originally was not designed for a modification of the Atlas after it was generated. Therefore, the LibVT has been modified by the implementation of two new functions, one function to delete a particular tile from VRAM, and one function to delete it from Tile Cache. Now, the graphic artist can edit a particular photograph, while

Figure 8: Visible artifacts while rendering in Scanopy resulting from simultaneous usage of the old (patterned) and the new version of tiles corresponding to a particular photograph.

Scanopy is running, and can start the update procedure by keystroke to see the changes inside the 3D model immediately.

We chose C++ as programming language to use a library J. Mayer proposed in his thesis [5]. This library, called libjpeg-turbo[6], produced the best results regarding loading JPEG-images from hard drive. Because of the large quantity of textures, the Atlas and the Tile Store provide, JPEG is because of its high compression rate the image format of choice. Like the application for mask generation, our program for the generation of the Atlas and the Tile Store can be called via "SendTo". The only file the application expects is a small text file with all configuration parameters. The most important of these parameters are the maximum side length of the Atlas parts, the paths where the Atlas and the Tile Store should be stored, the side length of the tiles as well as the output format of the Atlas and the Tile Store. The configuration file must be placed in the directory of the images, which should be contained in the Atlas. The application will consider all available image files in the base path of this text file. If all parameters are valid, the generation of the Atlas and the Tile Store starts. The results are the Atlas, split into as many parts, so that the desired maximum side length is not exceeded and the Tile Store, consisting of tiles with the desired side length.

5 Results

Our first application for the mask generation is already in use by the graphic artist and provides considerable ease of his work. In Figure 9 you can see the result of the mask generation for a single photograph, in Figure 10 the usage of the mask inside Adobe Photoshop is shown.

Our second application for the generation of the Atlas and Tile Store is significantly faster regarding Atlas gener-

[6]http://libjpeg-turbo.virtualgl.org/

Figure 9: Photograph used as texture (left side) with its corresponding mask (right side).

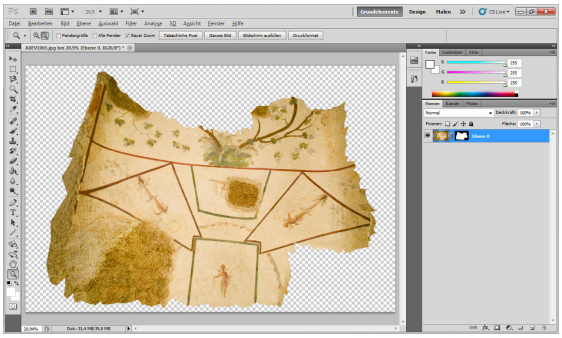


Figure 10: Usage of a generated mask inside Adobe Photoshop.

ation than the existing Python implementation, as you can see in Figures 11 and 12. Both diagrams show, that our application is at least four times faster than the Python script with ImageMagick. While our application produces nearly consistent results, the Python script needs the longer the more parts are produced. This increase can be explained by the nature of the script, which calls ImageMagick for every single image it produces. So when 64 Atlas parts are desired, ImageMagick must be called 64 times. This also means, that it has to read a particular photograph for every part, the photograph belongs to, again. In contrast to this behavior, our application holds as many photographs in memory as possible, to read them only once.

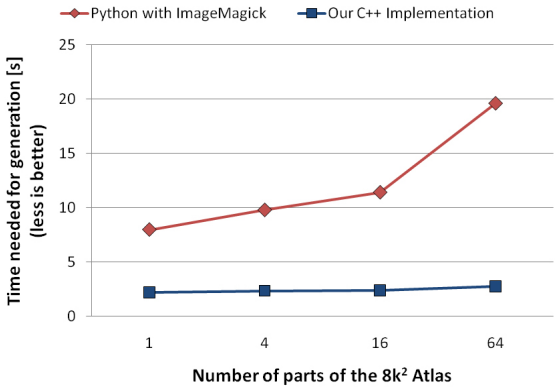


Figure 11: Performance of the generation of an $8k^2$ Atlas using our C++ implementation compared to the existing one.

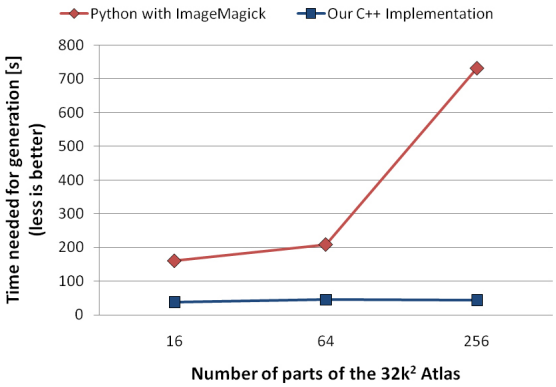


Figure 12: Performance of the generation of a $32k^2$ Atlas using our C++ application compared to the existing one.

The generation of the Tile Store delivers even more dramatic results. As you can see in Figures 13 and 14, our application is at least ten times faster than the Python script. To be consistent, the python script has been altered to use ImageMagick instead of the Python Imaging Library. Because of the huge time consumption of the Python script, the generation of the Tile Store with a tile resolution of 128^2 has been omitted for the $32k^2$ Atlas. With our application, this takes only 158.1 seconds. We also tested the generation of a $128k^2$ Atlas with its corresponding Tile Store with a tile resolution of 128^2. The Atlas was generated in 9min 11s, the Tile Store (11 levels with 1,398,101 tiles) in 2h 58min 21s.

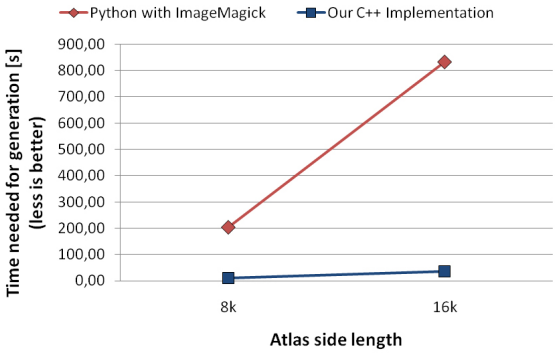


Figure 13: Performance of the generation of a Tile Store using our C++ implementation compared to the existing one. The tiles have a resolution of 128^2.

The update procedure again shows the speed-up when our application is used instead of a Python script. The Python script doing the update procedure was implemented to show a comparison. In Figure 15 you can see the times needed for an update.

The times for Atlas generation have been measured on a Hewlett Packard Pavilion dv6599eg notebook with an Intel Core2Duo T7300 processor with 2.0 GHz, 2 GB RAM and an nVidia GeForce 8400M GS graphics card. The times

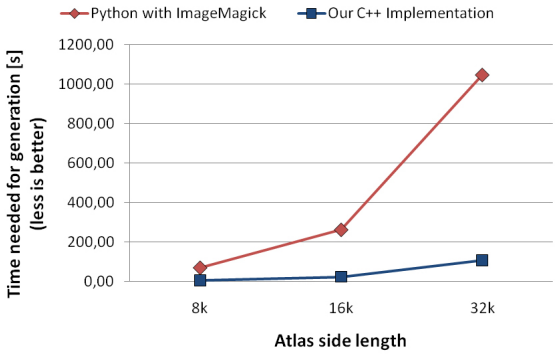


Figure 14: Performance of the generation of a Tile Store using our C++ implementation compared to the existing one. The tiles have a resolution of 256^2.

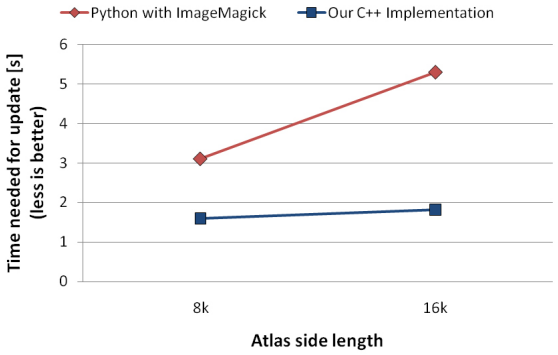


Figure 15: Performance of the update process using our C++ implementation compared to a Python script.

for Tile Store generation and update procedure have been measured on an Intel i7 2600k processor with 3.4 GHz, 8 GB RAM and an nVidia GeForce GTX 570. Only the 128k Atlas was also generated on the i7.

6 Conclusions and Future Work

We have presented an optimization of the workflow for a graphic artist, who is currently editing a large quantity of photographs used as textures for laser-scanned models. Our approach is based on the development of two applications and the introduction of Scanopy implementing Virtual Texturing into his workflow to ease his work. The first application is a mask generation program to visibly emphasize image areas corresponding to visible areas in the final model. Our second application is used for fast Atlas and Tile Store generation and update.

In future, the workflow could be further improved by transformation of the surrounding photographs into the plane of the actually edited photo. One approach to do this is the generation of a list with all photos and the corresponding neighbors, to know which of them have to be opened inside the editing program as a reference. This could decrease the number of checks inside the rendered 3D model.

Further, usage of the mask information for the Atlas and Tile Store generation might be useful to reduce the final size of the produced JPEG images, because of the higher compression rate when using input material with big homogeneous areas. An even smaller Atlas could be achieved by tightly packing of the visible image material. Therefore a transformation of the texture coordinates would be necessary, to reference the right areas inside this new Atlas.

A fully automated editing of the photographs corresponding to a model is desirable. This could be done with an approach based on Poisson Image Editing by Pérez, Gangnet and Blake [6]. There are already approaches for stitch-less image composition, that could be introduced to further ease the work of the graphic artist: [4], [3].

References

[1] Ahmed Abdelhafiz. *Integrating Digital Photogrammetry and Terrestrial Laser Scanning*. PhD thesis, Technical University Braunschweig, 2009.

[2] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia. *MeshLab: an open-source 3D mesh processing system*, April 2008.

[3] Ran Gal, Yonatan Wexler, Eyal Ofek, Hugues Hoppe, and Daniel Cohen-Or. Seamless montage for texturing models. *Comput. Graph. Forum*, pages 479–486, 2010.

[4] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless image stitching in the gradient domain. In *ECCV (4)*, pages 377–389, 2004.

[5] Albert Julian Mayer. Virtual texturing. Master's thesis, Vienna University of Technology, 2010.

[6] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 313–318, New York, NY, USA, 2003. ACM.

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

[8] Michael Wimmer and Claus Scheiblauer. Instant points. In *Proceedings Symposium on Point-Based Graphics 2006*, pages 129–136. Eurographics, Eurographics Association, July 2006.

Particle-based Visualization of Large Cosmological Datasets

Niko Lukač*
supervised by: Borut Žalik†

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

Abstract

Large quantities of simulated cosmological particle-based data cause considerable problems when it comes to real-time visualization. This paper considers an out-of-core approach for solving visualization problems on a single-desktop workstation. The approach proposed in this paper consists of two phases: the data preprocessing and its visualization. During the preprocessing, the cosmological data is hierarchically organized and efficiently ordered. Before rendering, the data is streamed to the memory from the disk. The culling techniques, such as view frustum culling and level-of-detail (LOD) are applied for visualization. In most cases, the real-time visualization of large cosmological particle datasets is achieved.

Keywords: Cosmological data, Particle-based visualization, Level-of-detail

1 Introduction

A computer based 3D visualization of observed or simulated particle data provides an insight for astrophysicists into a better understanding of the universe's properties. Cosmological visualization is extremely important for enabling the interactive exploration of not fully understood phenomena. Because the quantity of cosmological particle-based data increases every year, it has become a challenge to visualize this data in real-time on a single desktop workstation despite the recent advances in technology.

This paper presents an approach for the efficient visualization of large cosmological datasets using various paradigms, such as hierarchical data structure (octree), data ordering using hierarchical clustering, particle culling (frustum culling and LOD), data streaming, and prefetching.

The paper is organized into 6 sections. Section 2 provides a short summary and comparison with related work. Section 3 briefly describes cosmological particle-based data origin and properties. Section 4 focuses on the proposed visualization approach. Section 5 describes the results of our experiments on a desktop workstation. The conclusion summarizes the paper and suggests for improvements.

2 Related work

Several out-of-core point-based rendering approaches have already been developed over recent years [1], in order to efficiently visualize particle-based data. There are different kinds of hierarchical data structures, and LOD methods. Most of these methods are targeted towards rendering surface or mesh-based point-cloud data [2, 3, 4, 5]. The proposed method is designed to visualize cosmological particle-based data that can exceed system memory and represent an entirely different point distribution i. e. the "cosmic web". Because of clear differences in point distribution, the proposed method differs from point-cloud surface rendering methods, even though there are some similarities: use of clustering for LOD simplification [5] and hierarchical culling [2, 3, 4].

Several different approaches exist for cosmological particle-based visualization: advanced visualization toolkits [6], particle raytracing [7], isosurfaces extraction [8], parallel-based visualization using clusters of computers [9], particle culling methods [10, 11], spatial data division by hierarchical data structures [10 ,11, 12], exploiting the advantages of GPU [9, 10, 11], particle splatting techniques [11, 12], and particle attributes quantization and/or compression [11].

3 Cosmological particle data

Cosmological data is obtained from either observations or simulations. Most simulations are linked to the cosmic expansion of the universe [13, 14, 15] within a spatial cube, the size of which is defined in megaparsecs [Mpc] (1 Mpc is ~3.262×10^6 light years or ~3.08×10^{19} km). A time-step snapshot from cosmological simulation is defined using cosmological redshift z, to describe the

* niko.lukac@uni-mb.si
† zalik@uni-mb.si

specific time of expansion. There are different N-body algorithms (cosmological codes) [16] that are used to run a particle-based simulation. They are based on a theoretical model describing not fully understood cosmological phenomena, such as dark matter. The data from observations contains portions of the observable universe [17] and is considerably smaller in size than simulated data. It is evident that large simulated cosmological particle datasets contain millions, even billions of particles, which makes it difficult to visualize them in real-time. One of the better known cosmological simulations is Millennium simulation [13], based on a Cold Dark Matter (ΛCDM) model, and contains over 2160^3 particles resulting in large ~320 GB datasets for each single time-step snapshot. Individual particle of a cosmological simulation can represent different structures (e. g. dark matter fluid) and has a range of various attached attributes, depending on the simulation type. Most common attributes are particle floating point values of mass, spatial position and velocity.

4 Visualization approach

Our approach includes several methods aimed at reducing hardware load by rendering fewer particles, without visible loss of quality. The hardware load has the following constraints: GPU memory capacity, system memory capacity, disk capacity, disk latency, and a data transfer bandwidth between different components. Large-scale cosmological structures (e. g. filaments, clusters and halos) need to be preserved, in order to retain quality during the visualization. The proposed visualization solution is targeted towards static particle data, such as specific time-step snapshot of cosmological simulations.

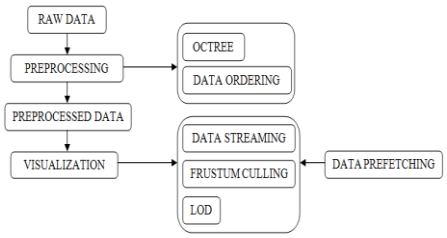


Figure 1: Workflow of the entire visualization approach.

The entire visualization process is shown in Figure 1. Firstly, the data is preprocessed by a hierarchical spatial subdivision using an octree, and efficient ordering. This phase is executed only once. Before rendering, the data has to be streamed from the disk to the system or, preferably, to the GPU memory. During the visualization phase, the rendering process is speeded-up by removing particles using view frustum culling, and LOD. Data prefetching is utilized to further increase the rendering performance.

4.1 Preprocessing phase

Firstly, the cosmological particle data is inserted into an octree structure. The octree is then constructed in a top-down manner, taking into account the particles positions. The inner nodes are only used for visibility testing, and the particles are stored inside the leaf nodes. The leaf's particles are stored on disk and later streamed to the GPU, because of the memory capacity constraint. A leaf node is constructed when there are fewer particles inside a given node than the threshold, which defines the maximum amount of particles that should be inside a leaf node at a specific tree depth. The threshold is defined as $1/(M–N+1)$ % of the number of particles from the node's parent node, where N is the node's tree depth and M is the maximum tree depth. When a node has more particles than the threshold, it will become an inner node and will be further divided. The particles inside a leaf node are spatially closer to the given leaf node's center than to any other node's center in the octree structure (see Figure 2). Although an inner node does not contain any data, it has information about the number of particles, which is the sum of the number of particles from all its descendants.

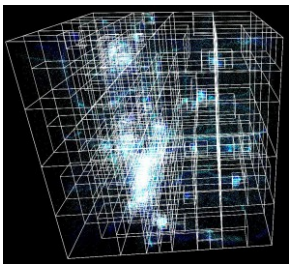


Figure 2: An example visualization of all octree nodes bounding boxes. The particles are visualized from all the viewable leaf nodes.

The adjacent leaf nodes are defined using the method described in [18] which is required for efficient data streaming and prefetching (see Subsections 4.5 and 4.6). At the end of preprocessing, the data inside each leaf node is additionally arranged using clustering, in order to minimize quality loss during LOD.

4.2 Data ordering

The order of the particle data inside a leaf node is the fundamental basis of the entire visualization quality. During the visualization phase, LOD calculates the amount of particles to be rendered for each visible leaf node. The data is read from the disk in linear order and streamed to the GPU, in order to gain the desired performance. Without careful data ordering, there would be huge quality losses during the visualization process, because of the removal of important particles.

The particle-based data inside a leaf is ordered according to the spatial distribution of the particles. A clustering algorithm is used that orders the particles using Euclidian distance metric. The results are particles that are bound in clusters according to their proximity. The smaller clusters (e. g. isolated particles) are more important than the larger clusters. The importance of a particle obtained in this way defines the particle order within the octree leaf node. Of course, the less important particles are retained, because they gain more importance

linearly as the camera moves closer. Consequently, at the end of the leaf node's final order, only the least important particles remain, belonging to the largest cluster.

As can be seen in Figure 3, the clusters of particles are arranged according to their capacities, in descending order. One particle from each arranged cluster is accepted in a bottom-up manner, and it is inserted into the final order, which is then stored on the disk for a particular leaf node.

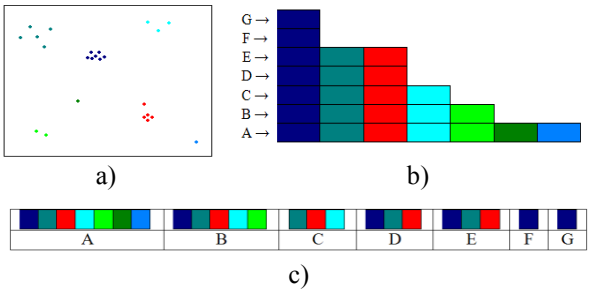


Figure 3: Data ordering - an illustrative example of particles data: a) colored clusters of particles, b) descending order of clusters according to capacities, c) final ordering of the particles within a stream.

There are several different and useful algorithms for spatial clustering [19]. The Chameleon hierarchical clustering algorithm, as proposed by Karypis et al. [20] was used in our implementation, because it offers a good trade-off between accuracy and speed.

4.3 Visualization phase

The visualization phase consists of three steps: particle culling, streaming, and rendering. Before rendering it is necessary to determine which particles to stream to the GPU. Frustum culling and LOD are employed for this purpose. The synergy of the culling and streaming allows the obtaining of suitable amount of data for rendering, based on the camera's viewpoint, and the view frustum's size. When the camera moves, any new data has to be streamed for rendering and the redundant data has to be removed from the memory. An illustration of the view frustum is shown in Figure 4.

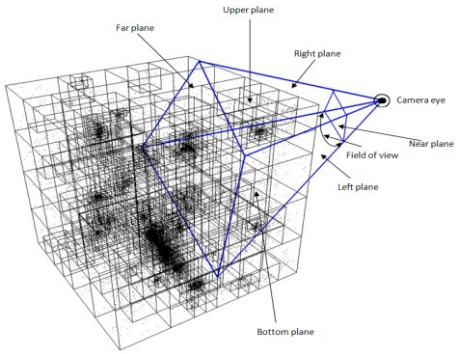


Figure 4: Illustration of the view frustum and the octree structure.

The view frustum culling and LOD can exploit the properties of the octree structure, because the data is preprocessed. The octree node's center points are used to represent the proximity of multiple particles within a specific spatial subvolume. Both culling methods traverse the octree structure from the root node and check the considered center points of the nodes for their visibility. Clearly, if the parent node is entirely outside the view frustum, its descendants are unchecked. The same applies if the entire node lies inside the view frustum. This allows for the skipping of many nodes during the culling process. The view frustum culling methods were used, as described in [21].

4.4 Level of Detail

After the view frustum culling is complete, the particles are further culled inside the view frustum using LOD. Better rendering performance is achieved, by adaptively adjusting the visualization details. The loss of quality is negligible, because the data was optimally ordered during the preprocessing phase. Consequently, the general particle distribution is preserved. The LOD streams the given leaf nodes data for rendering in linear order.

The LOD method uses the Euclidian distance metric between the center points of the leaf nodes and the position of the camera, in order to determine the amount of particles, to be loaded into the memory and then rendered for the given leaf node. The following equation is used for this:

$$\left(1 - \left(\frac{C_{dist}}{M_{dist}}\right)\right) * P_n \qquad (1)$$

where *Cdist* is the current distance from the camera to the leaf node, and *Mdist* is the maximum distance between the camera and the leaf node. In practice, this variable is limited by the distance from the front to the far plane of the view frustum. *Pn* is the total amount of particles in the leaf node. This equation provides similar effects as the traditional LOD error metric [1], which computes the projected screen space area of a leaf node.

The further the camera is from the considered leaf node, the fewer particles are rendered for this node. The direct consequence of this is that small details disappear from the greater distances, since larger clusters of particles are stuck within an area of few pixels or even a single pixel. The opposite happens, if the camera moves inside the given leaf node; in that case, all the particles inside the node are rendered. LOD quality can be observed in Figure 5, where the structure is retained intact even from the greater distances.

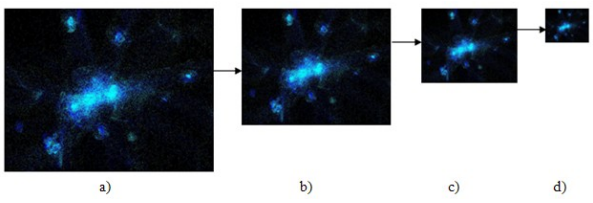


Figure 5: LOD on sample cosmological particle data (dark matter halo): a) 40%, b) 60%, c) 80% and d) 90% of *Mdist*.

4.5 Data streaming

The sizes of the large cosmological particle datasets exceed the memory capacity of any recent desktop workstation. The memory has to be used in an efficient manner, where only the recently viewed particles in a particular visualization instance are stored within the memory. Any preprocessed particle data from leaf nodes that have the same parent node are packed together and stored as a package into one file, in order to stream the data efficiently for thousands of octree leaf nodes from the disk to the memory. Threshold for a minimum package size has to be defined, because it is possible that a parent node does not contain 8 leaf nodes or that the capacities of the leaf nodes are small. In case the sum of the leaf nodes particles is below the threshold, the data of adjacent leaf nodes from other parent nodes is stored in the same file (see Figure 6). In our implementation the threshold was experimentally set to 1% of all particles for a given dataset. For a single snapshot dataset of the Millennium simulation, the minimum package file size is ~1.17 GB, storing only positions of the particles.

The reason for packing particle data together is based on the following premise: if one particular leaf node is rendered, then there is a high probability that the particles from the adjacent leaf nodes have be rendered, too. Prefetching is thus employed, at this stage.

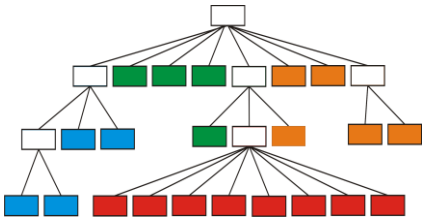


Figure 6: Example of non-balanced octree hierarchy, where the same colored leaf nodes are stored together.

4.6 Data prefetching

The use of available memory is maximized by prefetching proximal particle data outside the view frustum. A bounding sphere around the camera's center is constructed, for this purpose (see Figure 7).

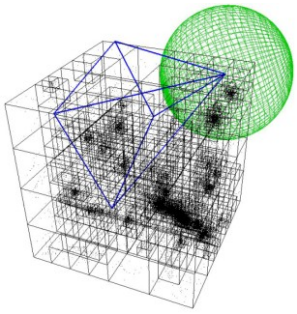


Figure 7: Illustration of the bounding sphere around the camera and the octree structure.

The radius of the bounding sphere is defined as the half distance between the front and the far plane of the frustum. This method is performed in the background and is efficient, when the camera is close to a specific area of interest. Based on the practical premise that the camera does not move over long distances when the area of interest is small, the required adjacent leaf nodes can be loaded into the memory. This applies to all the leaf nodes inside the defined bounding sphere or intersecting it, because the camera can move in any possible direction.

5 Results

This approach was tested on a single desktop workstation, using the following hardware: ATI Radeon HD 5750 GPU (1GB GDDR5 memory), AMD Phenom 1090T hexa-core CPU, 4 GB system memory (DDR3 1333 MHz), and SATA-II hard disk (7200 RPM, 1 TB capacity, 140MB/s average read speed). The OpenGL graphical library was used, where the particles were rendered as point primitive using color and alpha, without texture sprites. The streamed data from the disk to GPU was stored in VBO (Vertex Buffer Object) for each leaf. The visualization viewport resolution, was set at 1280x960, during the experiments.

In order to test the efficiency of the proposed visualization approach, several different and available cosmological particle simulation datasets were tested: Millennium (500 Mpc; z=0) [13, 24], MPA Larger box ΛCDM (479 Mpc; z=0) [23], GIF2 ΛCDM (110 Mpc; z=0) [14, 23], Mini Millennium (62.5 Mpc; z=0) [13, 24] and The Santa Barbara cluster (64 Mpc; z=0; gadget output) [15, 22].

The preprocessing required double the capacity of the input dataset. Around 80% of the preprocessing time was dedicated to data ordering for each dataset. However, this still took considerably less time than running a cosmological simulation, which takes several days on parallelized systems [13].

After preprocessing the quality loss and the performances gained for different datasets, were tested. To measure these properties, a predetermined camera path was made, which does a fly-through in the visualized data. The camera position was initially aligned with the X coordinate axis and distanced by the *Mdist* variable from the LOD. The cameras viewpoint was set to face the center point of the octree root node. The predetermined fly-through consisted of 10 steps. For each step the camera made a full 360 degree rotation, using 1 degree steps around the center point, and afterwards moved towards the center point over 10% of the initial distance *Mdist*. The testing was completed when the camera's position was equal to the center point. The speed was measured by performing fly-through three times for each dataset, in order to obtain more reliable results. FPS (Frames per second) was measured for speed comparison. A modified fly-through was made, in order to measure the quality difference. Each frame was visualized with the LOD both enabled and disabled. The per-pixel difference from both frames (LOD on and off) was calculated using the Euclidian distance metric, in

order to measure the quality. This comparison is suitable, because the projected particles on the frustum's near plane are the end result of rendering. The experimental results of the average culled particles for the whole fly-through, average fps and average quality decrease when LOD is on, are shown in Table 1.

Table 1: Results of the experiments on different cosmological datasets.

Cosmological particle-based dataset	Number of particles	Average culled particles	Average FPS	Average quality decrease
Millennium	10 077 696 000	88.2%	11.3	7.4%
MPA ΛCDM	134 217 728	80.2%	67.4	6.7%
GIF2 ΛCDM	64 000 000	85.3%	88.5	4.9%
Mini Mill.	19 683 000	87.9%	142.0	6.1%
The Santa Barbara	16 777 216	86.7 %	177.1	5.3%

Proposed out-of-core visualization approach was not compared to brute-force rendering, because this is practically impossible. A desktop workstation would run out of memory, when trying to render huge amount of particles (exceeding GPU and system memory several times) using the brute-force rendering approach.

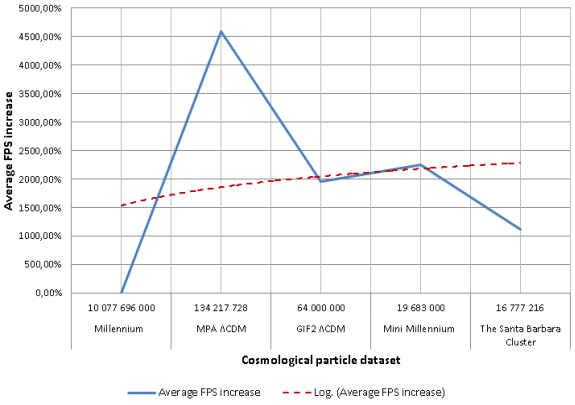


Figure 8: Average FPS increase (%).

FPS increased when employing LOD, providing a significant boost (see Figure 8). The quality decreased only slightly when the number of particles increased significantly (see Figure 9). When LOD was disabled, visualization remained in real-time, as long as there was enough GPU memory.

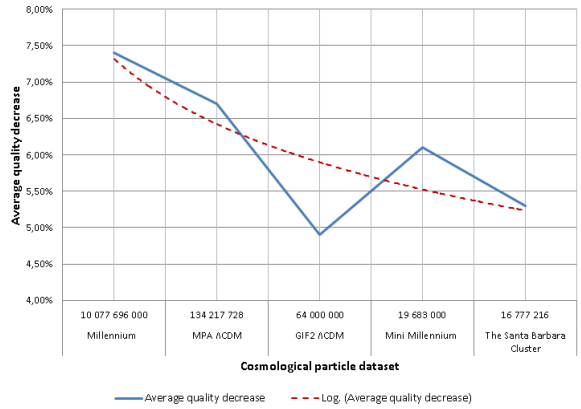


Figure 9: Average quality decrease (%).

Due to efficient data ordering and the hierarchical spatial subdivision, the average quality loss, which was around 5%, was hardly noticeable with the naked eye, as shown in the example in Figure 10. This is because the data ordering was done during the preprocessing phase, where the most clustered particles are of least importance (see Figure 10c).

a) b) c)

Figure 10: Sample visualized cosmological particle data using a) LOD enabled, b) LOD disabled and c) the difference (~5%).

Table 2 presents the averaged results for each even fly-through step for one of the largest tested particle dataset; Millennium. The average FPS dropped and the quality increased when the camera moved into the region of the visualized data. Within the areas of interest, LOD had less importance than frustum culling, which helped to preserve real-time visualization, since more nodes were being culled outside the view frustum, and data prefetching was being utilized in the background.

Table 2: Results for camera fly-through even steps of the Millennium dataset.

Step	Camera distance from octree root node center point (% of *Mdist*)	Average culled particles	Average FPS	Average quality decrease
2.	80%	96.9%	15.3	2.9%
4.	60%	84.2%	8.1	6.4%
6.	40%	88.5%	9.2	8.1%
8.	20%	94.8%	11.9	5.9%
10.	0%	95.3%	14.2	2.4%

Data prefetching was implemented on two background threads, each on its own CPU core, in order to have a smaller impact on the actual visualization. The FPS difference was measured by applying prefetching, using the same fly-through as before on the Millennium particle dataset. The results of this test are shown in Figure 11. The method paid off when the camera was close or inside the region of the visualized data.

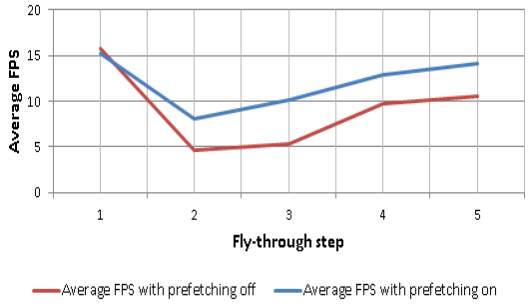


Figure 11: Average FPS for each camera fly-through even step of the largest tested dataset, with and without prefetching.

6 Conclusion

This paper shows that the large-structures of the cosmological particles are visualized in high details, with minimal impact from particle culling. We can efficiently render any cosmological particle dataset and in most cases achieve real-time visualization on a single desktop workstation, with a small quality loss.

We found that the main hardware bottlenecks on the desktop workstations are the disk read speed and on-demand data transfer from the memory to GPU in large quantities. In the near future such bottle-necks would limit real-time visualization, if the cosmological particle datasets snapshots are of the size of several terabytes. In order to overcome this problem, the presented approach could be extended using parallelization. There are also other ways for possible extension, such as data compression via GPU using technologies such as CUDA.

Acknowledgements

Some of the used datasets in this paper are from simulations carried out by the Virgo Supercomputing Consortium [23]. We would like to thank V. Springel et al. [13] for giving us insights in to the Millennium simulation, L. Gao et al. [14] into the GIF2 simulation and K. Heitmann et al. [15] into the Santa Barbara cluster simulation. The Millennium Simulation databases used in this paper and the web application providing online access to them were constructed as part of the activities of the German Astrophysical Virtual Observatory (GAVO) [24]. Thanks to Gerard Lemson from GAVO for his support on the transmission of large Millennium simulation raw particle data. Thanks to Mel Krokos from the University of Portsmouth, U.K., who initiated this work.

References

[1] M. Sainz, R. Pajarola, Point-based rendering techniques, ACM Computers and Graphics, vol. 28, no. 6, pp. 869-879, December 2004.

[2] C. Erikson, D. Manocha, W. V. Baxter III, HLODs for Faster Display of Large Static and Dynamic Environments, Proceedings of ACM Symposium on Interactive 3D graphics, New York, USA, March 2001.

[3] C. Dachsbacher, C. Vogelgsang, M. Stamminger, Sequential point trees, ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2003, vol. 22, no. 3, pp. 657~662, New York, USA, July 2003.

[4] S. Rusinkiewicz, M. Levoy, QSplat: A Multiresolution Point Rendering System for Large Meshes, ACM SIGGRAPH Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 343-352, New York, USA, July 2000.

[5] M. Pauly, M. Gross, L. P. Kobbelt, Efficient Simplification of Point-Sampled Surfaces, IEEE Visualization Proceedings, pp. 163-170, Boston, USA, November 2002.

[6] M. Comparato, U. Becciani, A. Costa, B. Larsson, B. Garilli, C. Gheller, J. Taylor, Visualization, Exploration and Data Analysis of Complex Astrophysical Data, The Publications of the Astronomical Society of the Pacific, vol. 119, no. 858, pp. 898-913, August 2008.

[7] K. Dolag, M. Reinecke, C. Gheller, S. Imboden, Splotch: Visualizing Cosmological Simulations, New Journal of Physics, vol. 10, no. 12, 2008.

[8] P. A. Navrátil, J. L. Johnson, V. Bromm, Visualization of Cosmological Particle-Based Datasets, IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 6, pp. 1712-1718, 2007.

[9] Z. Jin, M. Krokos, M. Rivi, C. Gheller, K. Dolag, M. Reinecke, High-performance astrophysical visualization using Splotch, Procedia Computer Science, ICS 2010, vol. 1, no. 1, pp. 1769-1778, May 2010.

[10] T. Szalay, V. Springel, G. Lemson, GPU-Based Interactive Visualization of Billion Point Cosmological Simulations, November 2008.

[11] R. Fraderich, J. Schneider, R. Westermann, Exploring the Millennium Run-Scalable Rendering of Large-Scale Cosmological Datasets, IEEE Transactions Visualization and Computer Graphics, pp. 1251-1258, June 2009.

[12] M. Hopf, T. Ertl, Hierarchical Splatting of Scattered Data, IEEE Visualization Proceedings, pp. 433-440, Washington, USA, October 2003.

[13] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, F. Pearce, Simulating the joint evolution of quasars, galaxies and their large-scale distribution, Nature, vol. 435, pp. 629–636, 2005.

[14] L. Gao, S. D. M. White, A. Jenkins, F. Stoehr, V. Springel, The subhalo population of LCDM dark haloes, Monthly Notices of the Royal Astronomical Society, vol. 355, no. 3, pp. 819-823, 2004.

[15] K. Heitmann, P. M. Ricker, M. S. Warren, S. Habib, Robustness of Cosmological Simulations I: Large Scale Structure, The Astrophysical Journal Supplement Series, vol. 160, no. 1, pp. 28-58, 2005.

[16] K. Heitmann, Z. Lukic, P. Fasel, S. Habib, M. Warren, M. White, J. Ahrens, L. Ankeny, R. Armstrong, B. O'Shea, P. M. Ricker, V. Springel, J. Stadel, H. Trac, The Cosmic Code Comparison Project, Computational Science and Discovery, vol. 1, no. 1, 2008.

[17] J. R. Gott III, M. Juric, D. Schlegel, F. Hoyle, M. Vogeley, M. Tegmark, N. Bahcall, J. Brinkmann, A Map of the Universe, The Astrophysical Journal, vol. 624, no. 2, pp. 463-484, 2005.

[18] J. Vörös, A strategy for repetitive neighbor finding in octree representations, Image and Vision Computing, vol. 18, no. 14, pp. 1085-1091, 2000.

[19] J. Han, M. Kamber, A. K. H. Tung., Spatial Clustering Methods in Data Mining: A Survey, Geographic Data Mining and Knowledge Discovery, vol. 21, pp. 1–29, 2001.

[20] G. Karypis, E. Han , V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer, vol. 32, no. 8, pp. 68-75, 1999.

[21] U. Assarsson, T. Möller, Optimized view frustum culling algorithms for bounding boxes, Journal of Graphics Tools, vol. 5, no. 1, 2000.

[22] The Cosmic Data Bank, Online: http://t8web.lanl.gov/people/heitmann/arxiv/data.html (14.03.2008)

[23] MPA Numerical Cosmology, Online: http://www.mpa-garching.mpg.de/NumCos/ (30.10.2006)

[24] German Astrophysical Virtual Observatory (GAVO) – Virgo Millennium Database, Online: http://www.g-vo.org/Millennium/ (10.02.2011)

Order Independent Transparency with Per-Pixel Linked Lists

Pál Barta*
Balázs Kovács†
Supervised by: László Szécsi‡ and László Szirmay-Kalos§

Budapest University of Technology and Economics
Budapest / Hungary

Abstract

This paper proposes a method for rendering scenes of both opaque and transparent objects. Transparency depends on the attenuation coefficient and the thickness of the transparent object we wish to render. To get the visible radiance, the volume rendering equation should be solved. Instead of marching a ray, we build a list that contains the intersection points of the ray and object surfaces. In the second phase of rendering, the GPU sorts and processes the lists and evaluates the attenuation integrals analytically, considering also the order of the segments. This solution is mathematically correct even if objects intersect, i.e. it does not involve drastic simplifications, and provides high framerates even on moderately complex scenes, outperforming previous methods. In addition to transparent objects, the technique is also appropriate to visualize natural phenomena represented by particle systems.

Keywords: Transparency, Direct3D 11, Linked-lists, GPU, Particle Systems.

1 Introduction

The technique of alpha blending has a long history in two- and three-dimensional image synthesis. There are many ways to blend colors [10], but the most important issue is that we can only get realistic results if we sort transparent objects by their distance from the camera. Unfortunately, this requirement is not compatible with incremental rendering and z-buffer based visibility determination, which allow the processing of objects in an arbitrary order. Sorting objects or even triangles in a way that occluders follow objects occluded by them is difficult and is usually impossible without further subdivision of objects. The problem is that an object is associated with a depth interval and not with a single distance value, so no direct ordering relation can be established. A possible solution for non-intersecting triangles is the application of the painters algorithm [1], but this has super-linear complexity and its GPU implementation is prohibitively complicated.

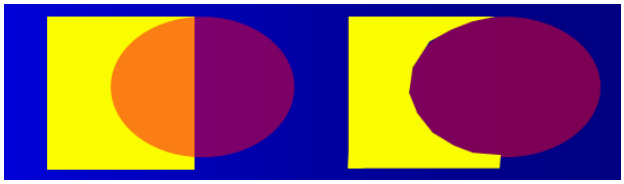


Figure 1: Order matters when the scene contains transparent objects.

If objects may intersect each other, then the situation gets even worse. A typical case of intersecting transparent objects are particle systems, which are tools to discretize, simulate and visualize natural phenomena like fog, smoke, fire, cloud, etc. The simplest way of their visualization applies planar billboards, but this approach results in abrupt changes where particles intersect opaque objects. The solution for this problem is the consideration of the spherical extent of the particle during rendering, as proposed in the concept of *spherical billboards* [9], also called *soft particles*. Spherical billboards nicely eliminate billboard clipping and popping artifacts at a negligible additional computational cost, but they may still create artifacts where particles intersect each other. Most importantly, when the z-order of billboards changes due to the motion of the particle system or the camera, popping occurs. This effect is more pronounced if particles have non-identical colors or textures.

Instead of executing the sorting for the objects, we can as well ensure the correct order on the level of fragments. This approach does not require the sorting of the objects on the CPU, which is emphasized by its name, *order independent transparency*. The family of such methods is usually referred to as *depth peeling*. The basic idea of depth peeling is that the fragment shader may discard fragments that are not farther than a previously selected threshold and the depth buffer will identify the closest fragment from the not discarded points. Thus, the scene is rendered multiple times and each time we ignore the already identified layers. Intuitively, we peel layer surfaces from the scene. Depth peeling has been used in global radiosity [6] and in transparency [3, 8] calculation as well. Unfortunately, depth peeling needs to render the scene multiple times, de-

*brazil.hu@gmail.com
†kockafely@gmail.com
‡szecsi@iit.bme.hu
§szirmay@iit.bme.hu

pending on the *depth complexity* of the scene (the depth complexity is defined as the maximum number of intersections a ray has in a given scene). Even its advanced versions, like *Dual Depth Peeling* [5], *Reverse Depth Peeling* [8] etc. could not be used effectively in real-time scenes without limiting the sorting to just the first few layers.

AMD presented a special solution at the Game Developers Conference 2010 [4]. The latest series of ATI Radeon supports DirectX11, which opened the door for new rendering algorithms. The latest Shader Model 5.0 GPUs have new features like the *read/write structured buffers* or the *atomic operations* to manipulate them. Recall that before Shader Model 5.0, shaders may either read memory (e.g. input textures) or write it (e.g. render target), but not both, and writes are always exclusive, so no synchronization is necessary. This limitation has been lifted by Shader Model 5.0, and now we do not have to wait for the end of a pass before reading back the result in a shader processor. With the use of these features, we are able to process the incoming fragments in a complex way instead of writing them to the frame buffer. The fragments can be stored in linked lists, which creates new ways to implement order-independent alpha-blending.

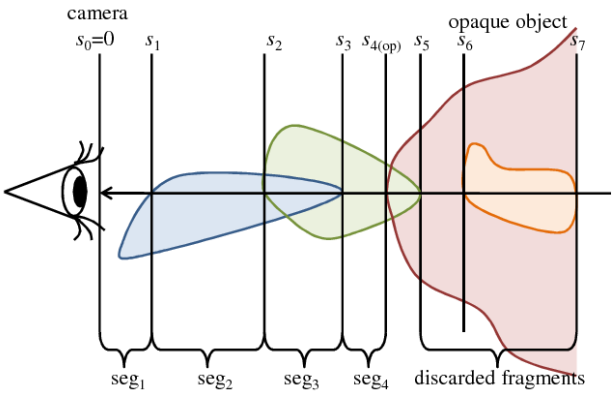


Figure 2: Model of a scene that contains opaque and transparent objects. Each transparent object is homogeneous and they can intersect each other.

This paper proposes a new algorithm to render transparent, possibly intersecting objects and particles based on DirectX11's structured buffers and atomic operations. In Section 2, we first survey the model of light transport in homogeneous transparent objects. Then, in Section 3 the new, GPU-based algorithm is discussed. Finally, we present results and conclusions.

2 Model of light transport in homogeneous objects

In case of scenes having only opaque objects the radiance is constant along a ray and scattering may occur just on object surfaces. Participating media, however, may scatter light not only on their boundary, but anywhere inside their volume. Participating media can be imagined as some material that does not completely fill the space. Thus the photons have the chance to go into the media and to travel a random distance before collision. To describe light–volume interaction, the basic rendering equation should be extended [7, 2]. The volumetric rendering equation is obtained considering how the light goes through participating media (Figure 3).

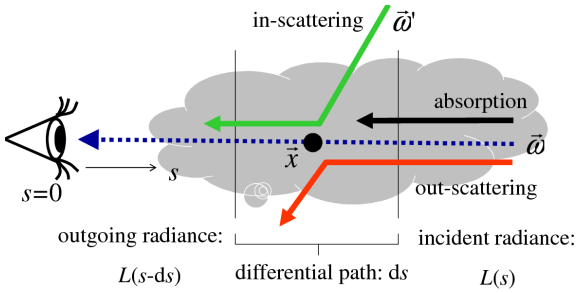


Figure 3: Change of radiance in participating media.

The change of radiance L on a path of differential length ds and of direction $\vec{\omega}$ depends on different phenomena:

Absorption and out-scattering: Photons may collide with the material and the material may or may not reflect the photon after collision. The intensity change is proportional to the number of photons entering the path, i.e. the radiance and the probability of collision. If the probability of collision in a unit distance is τ, then the probability of collision along infinitesimal distance ds is τds. After collision the particle is reflected with the probability of *albedo* a, and absorbed with probability $1-a$. Collision density τ and the albedo may also depend on the wavelength of the light. Summarizing, the total radiance change due to absorption and out-scattering is $-\tau L\text{d}s$.

In-scattering: Photons originally flying in a different direction may be scattered into the considered direction. The expected number of scattered photons from differential solid angle dω' equals to the product of the number of incoming photons and the probability that the photon is scattered from dω' to $\vec{\omega}$ in distance ds. The scattering probability is the product of the collision probability (τds), the probability of not absorbing the photon (a), and the probability density of the reflection direction $\vec{\omega}$, given that the photon arrived from direction $\vec{\omega}'$, which is called *phase function* $P(\omega', \omega)$. Following an ambient lighting model, we assume that the incident radiance is L^a in all directions and at every point of the scene. Taking into account all incoming directions Ω', the radiance in-

crease due to in-scattering is:

$$\tau a \mathrm{d}s \left(\int_{\Omega'} L^a P(\omega', \omega) \mathrm{d}\omega' \right) = \tau a L^a \mathrm{d}s$$

since the phase function is a probability density, thus its integral over the full directional domain equals to 1.

Adding the discussed changes, we obtain the following *volumetric rendering equation* for the radiance L of a ray at $s - \mathrm{d}s$ having taken step $\mathrm{d}s$ toward the eye:

$$L(s - \mathrm{d}s, \vec{\omega}) = (1 - \tau \mathrm{d}s) L(s, \vec{\omega}) + \tau a L^a \mathrm{d}s. \quad (1)$$

Subtracting $L(s)$ from both sides and dividing the equation by $\mathrm{d}s$, the volumetric rendering equation becomes a differential equation.

$$-\frac{\mathrm{d}L(s, \vec{\omega})}{\mathrm{d}s} = -\tau L(s, \vec{\omega}) + \tau a L^a. \quad (2)$$

In homogeneous media, volume properties τ and a are constant. In our model, the scene contains homogeneous objects having different materials. Thus, in our case, the properties are piece-wise constant functions along a ray.

2.1 Solution of the simplified volumetric equation

The radiance along a ray is described by an inhomogeneous first-order linear differential equation, which can be solved analytically. Assuming that the background radiance is zero, the radiance at the eye position ($s = 0$) is:

$$L(0, \vec{\omega}) = \int_0^\infty \tau(s) a(s) L^a e^{-\int_0^s \tau(x) \mathrm{d}x} \mathrm{d}s$$

where direction $\vec{\omega}$ points from the pixel towards the eye. Let us now exploit the fact that material properties $a(s)$ and $\tau(s)$ are piece-wise constant functions, they may change where the ray intersects the surface of an object. Let us denote the distance values of ray surface intersections by $s_1, s_2, \ldots, s_n$ and extend this ordered list by $s_0 = 0$ and $s_{n+1} = \infty$. The ray, i.e. the domain of the integration is partitioned according to the segments between the intersection points, where albedo $a(s)$ and attenuation parameter $\tau(s)$ are equal to a_i and τ_i in segment $[s_{i-1}, s_i)$, respectively:

$$L(0, \vec{\omega}) = \sum_{i=1}^{n+1} \int_{s_{i-1}}^{s_i} \tau_i a_i L^a e^{-\int_0^s \tau(x) \mathrm{d}x} \mathrm{d}s.$$

Then, we also partition the $[0, s]$ interval according to the intersection points in the attenuation formula, assuming that s is in $[s_{i-1}, s_i]$:

$$\int_0^s \tau(x) \mathrm{d}x = \tau_i (s - s_{i-1}) + \sum_{j=1}^{i-1} \tau_j (s_j - s_{j-1}).$$

Thus, the exponential decay is:

$$e^{-\int_0^s \tau(x) \mathrm{d}x} = e^{-\tau_i (s - s_{i-1})} \prod_{j=1}^{i-1} e^{-\tau_j (s_j - s_{j-1})}.$$

Substituting this back into the eye radiance, we obtain:

$$L(0, \vec{\omega}) = \sum_{i=1}^{n+1} a_i L^a \int_{s_{i-1}}^{s_i} \tau_i e^{-\tau_i (s - s_{i-1})} \mathrm{d}s \prod_{j=1}^{i-1} e^{-\tau_j (s_j - s_{j-1})}.$$

We can obtain better quality rendering with shading without significantly slowing down the rendering. We are using Rayleigh shading where the phase function is:

$$P(\cos\theta) = \frac{3}{16\pi} (1 + \cos\theta^2)$$

θ is the angle between the light and the view direction. For simplicity we assume the light source is directional, thus we can foil the Rayleigh term before the integral because θ will be constant in a segment. Therefore we can substitute L^a in our equations by

$$L^s = L^a + L^r \frac{3}{16\pi} (1 + \cos\theta^2).$$

We introduce a shorthand notation for the *color contribution* C_i of a segment:

$$C_i = a_i L^s \int_{s_{i-1}}^{s_i} \tau_i e^{-\tau_i (s - s_{i-1})} \mathrm{d}s = a_i L^s \left(1 - e^{-\tau_i (s_i - s_{i-1})} \right). \quad (3)$$

Note that this formula remains valid also for the $\tau_i = 0$ case, i.e. when the ray travels in free space.

Similarly to the contribution, *segment transparency* T_i can also be applied to segment i

$$T_i = e^{-\tau_j (s_{j+1} - s_j)}. \quad (4)$$

With these shorthand notations, the radiance associated with a particular pixel is

$$L(0, \vec{\omega}) = \sum_{i=1}^{n+1} \left(C_i \prod_{j=1}^{i-1} T_j \right). \quad (5)$$

The evaluation of this formula requires the intersection points to be sorted and segments to be visited in the sorted order. If the distances are sorted in ascending order, at each segment two equations should be evaluated iteratively, starting with $L = 0$ and $T = 1$:

$$\begin{aligned} L &\leftarrow L + C_i T, \\ T &\leftarrow T \cdot T_i. \end{aligned}$$

When the last segment is processed, variable L contains the radiance of the ray.

There are two critical issues concerning the iterative evaluation of these formulae. First, segments should be

stored and sorted on the GPU, without knowing in advance how many segments a particular ray has. On the other hand, the properties of the segments, including the albedo and the attenuation parameter, should be determined from the object properties. As objects may intersect each other, this question cannot be simply answered by checking whose surface the ray has most recently crossed. These problems are solved by the algorithm presented in the next section.

3 The rendering algorithm

The algorithm consists of two main steps: At first we collect every intersection point between all rays and object surfaces in lists associated with rays. Then, in the second phase, the lists are sorted and processed to get the pixel colors.

3.1 Data structures

In order to get not only the first ray surface intersection, but all intersections of an eye ray, the latest features of the DirectX 11 compatible GPUs are exploited, including structured buffers and atomic writes.

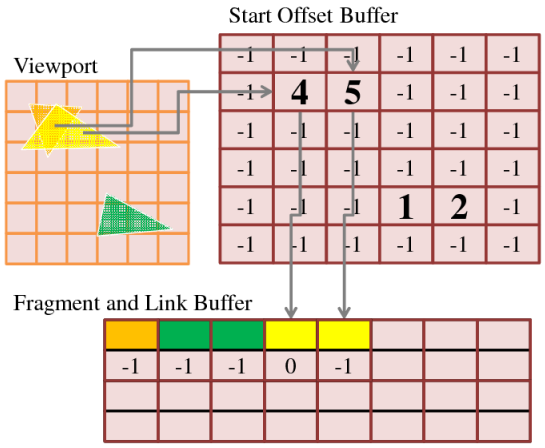


Figure 4: Data structures.

In particular, we use types of "Byte Address Buffer" and "Read/Write Structured Buffer". The first type is used as the starting element of the linked lists, while the buffer of the other type is filled with the fragment data structure. We need also a texture containing the opaque scene with depth information. The main data stores are the "Fragment and Link Buffer", the "Start Offset Buffer", and the "Opaque Scene Texture" (Figure 5).

Fragment and Link Buffer: The output data of the first-phase pixel shader contains the radiance reflected at the surface, the volumetric attenuation coefficient and the albedo of the object, a flag whether or not the surface is front facing, and the distance of the fragment from the eye. We would like to store these values in a structure and build a list of them for each pixel. Instead of writing data to the frame buffer, we have to store them in a special buffer called "Read/Write Structured Buffer". It is a generic buffer that contains the declared type of structure. We have to append a pointer to each structure, which addresses the next fragment in the list. The value -1 as address denotes the end of the list. The size of the buffer depends on the estimated amount of transparent fragments in the viewing frustum. If the allocated memory is not enough for all the transparent fragments, then it will overflow and we will lose important data. We set a counter for the buffer, its initial value is 0. When the shader appends a new structure of fragment data to the buffer, the value of the counter will provide the address of the new element. Afterwards we increment the counter, so it always addresses the next free slot in the Fragment and Link Buffer. We should update the counter with atomic operators, because it is parallely used by a number of shader units.

Start Offset Buffer: The type of this buffer is the "Byte Address Buffer", which holds 4-byte addresses. The function of this data structure is to refer to the first element of the linked list for every pixel. The first element is always the last fragment that was processed for that list. When a new structure is appended, the pointer of the structure will get the value of the associated element in the "Start Offset Buffer". Accordingly, we write the address of the newly stored structure in the buffer. We have to allocate memory to store one address for each pixel on the viewport.

Opaque Scene Texture: This shader resource holds the RGB values of the opaque scene, and the distance of the fragment from the eye, which is stored in the alpha channel. When processing the list, the color read from this texture is also blended and the fragments being farther than this opaque fragment are discarded.

3.2 Collecting the fragments

The method starts with rendering opaque objects. The alpha channel is used to store the distance of the fragment from the camera, so later the list of fragments can be cropped based on this value.

The next step is to render the transparent objects. The drawing function sets up the pipeline and passes the vertex information to the GPU. It is important that the culling of the back-facing fragments must be disabled, otherwise the GPU discards them and lot of important data about the transparent scene will be lost. The vertex shader performs the standard transformations, the novel part of the rendering comes with the pixel shader. When the pixel shader gets a fragment as input, it allocates and fills a structure, which contains the surface radiance, the volumetric attenuation and albedo, the distance from the cam-

era, and the orientation of the fragments. Then the shader stores the structure in the "Fragment and Link Buffer" at the next free slot and sets the "Start Offset Buffer" using the screen coordinates of the currently processed fragment. The pointer in the "Start Offset Buffer" at the address of the screen coordinate will reference this structure, and the pointer of the newly inserted structure gets the former value of the address buffer. This way a new element is inserted into the linked list.

3.3 Loading and sorting the lists

After the lists are created for each pixel and all the visible fragment information is in the memory of the GPU, the second phase of the rendering begins. The next step is to parse each list in a local buffer, then sort the elements in ascending order of the distance from the camera. Loading the lists into the local buffer of the shaders at first is important because working with local data is easier and faster than reading each structure separately from a shared resource. To avoid copying larger set of data, an index buffer should also be created, so during the sort the GPU moves only indices instead of structures.

3.4 The sorting

Every sorting algorithm has its own characteristic, which offers advantages and disadvantages depending on the environment parameters of the sorting. These algorithms are based on different concepts like partitioning, merging, selection or insertion, some of them providing bigger overheads or different efficiency under different conditions. The implementation of the methods can also have a slight effect on their performance. At first we have to give an estimated number of the elements of the array we have to sort. In our case the fragments for each pixel are stored in these arrays, the GPU will sort each array independently and will run the same sorting algorithm for each pixel. So we have one array of fragments in our shader code, which belongs to the currently processed pixel. Most of the cases the rendered scene contains some transparent objects in an opaque environment, but even if we are experimenting with more complex composition, it is reasonable to say that the average depth complexity of a transparent scene is lower than 100 layers. The algorithms have average, worst and best case scenarios, since the GPU will run the sorting many thousand times for each frame, we have to analyze the average speed of the method. Another important attribute to consider is whether the elements are presorted in some way. We can assume here, that the fragments are collected randomly. However, since we have information about the order of the objects drawn to the screen, some kind of presorting could be possible, but right now this enhancement is left for the future. The computational complexity is maybe the most important parameter of an algorithm along with memory usage, we try to find a sorting algorithm that performs the best under the above mentioned conditions and needs the lowest amount of memory space. Quicksort is one of the most popular sorting algorithms and its optimized versions are considered the fastest of them. As a general rule this statement can be true, however, it is a bit more complex than the simpler methods, which provides remarkable overhead. Additionally the current GPUs do not support recursive calls, and the basic quicksort is a recursive algorithm. Its non-recursive version needs also a stack to implement generating more overhead. In the case of relatively small lists a much simpler algorithm can perform better and the insertion sort is a popular choice. Its disadvantages begin to come forward on longer lists, while sorting an array of 100 items the insertion sort outperforms the other sorting algorithms. (http://warp.povusers.org/SortComparison/integers.html) Additionally its easy to implement, does not change the relative order of elements with equal keys, only requires a constant amount of additional memory space and performs even more faster on pre-sorted lists. These attributes makes us insertion sort the best choice here.

3.5 Processing the lists

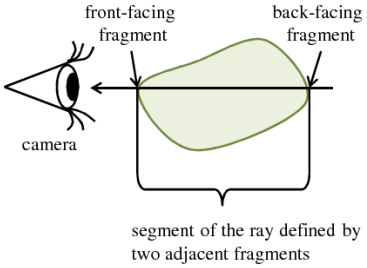


Figure 5: Distinction of front and back facing surfaces.

After the sorting, the shader blends the fragments according to the physical model of light absorption, and every segment yields its contribution in the final value. When the GPU finishes sorting the list, the sorted array of fragments divides the ray into *segments*. In our model, both the intersecting and the stand-alone objects divide the space into homogenous regions, which are represented by the segments defined by the list of the fragments. Each segment possesses properties of contributed color C_i and transparency T_i (equations 3 and 4). These values are based on the attributes of the objects and the length between consecutive intersections.

If objects intersect, the contributed color and the transparency should be computed from the length the ray travels in the intersection and the combined attenuation and albedo. Suppose that the attenuation coefficients are τ_1 and τ_2 in object 1 and object 2, respectively, and similarly their albedos are a_1 and a_2. In their intersection, the particles of both objects are present, so the probability density of photon–particle collision is the sum of the elementary

probability densities:

$$\tau = \tau_1 + \tau_2.$$

Considering the albedo, i.e. the conditional probability that reflection happens given that the photon collides, we can obtain the following combined value:

$$a = \frac{\tau_1 a_1}{\tau_1 + \tau_2} + \frac{\tau_2 a_2}{\tau_1 + \tau_2}.$$

The properties of the segments can be determined while traversing the list. The list items represent surfaces, i.e. object boundaries, where the optical properties of only the respective object can be fetched. Thus, the optical properties of the segments, including the combination of the albedos and the attenuation coefficients, need to be computed on the fly while we are traversing the list.

Suppose, we maintain two running variables a and τ that represent the albedo and the attenuation coefficient of the current segment. When a list item is processed, i.e. when we cross a surface, these running variables are updated. If the list item represents a front facing surface, then we enter object o whose parameters are stored in the list, in variables a_o and τ_o. Consequently, the running albedo and attenuation are updated to reflect the properties of this new object:

$$\begin{aligned} a &\leftarrow (\tau a + \tau_o a_o)/(\tau + \tau_o), \\ \tau &\leftarrow \tau + \tau_o. \end{aligned}$$

When the traversal encounters a back-facing fragment, we leave an object, thus its attenuation and albedo should be removed from the combined values. To execute this, the inverse of the previous formulae should be evaluated:

$$\begin{aligned} \tau &\leftarrow \tau - \tau_o, \\ a &\leftarrow (\tau a + \tau_o a - \tau_o a_o)/\tau. \end{aligned}$$

In order to initialize these iterations when we start the list traversal from the eye position, we should determine which objects contain the eye position. This information can be found by traversing the ordered list backward starting at the farthest intersection point, where we assume that we came from free space, thus $\tau = 0$ and $a = 1$. During this initial traversal, the same operations are performed, just the roles of front-facing and back-facing segments are exchanged.

4 Results

The presented algorithm has been implemented in a DirectX 11/HLSL environment on an ATI Radeon 5700 graphics card. The modeled scene consists of 50000 opaque triangles and 150000 transparent triangles, the resolution for the tests is set to 800×600. The frame rate mainly depends on the depth complexity of the currently rendered scene. If no transparent object is present, the performance is about 110 FPS because of the overhead of the two-step process. As the depth complexity and the transparent area grow, the frame rate decreases. If there are more than 20-30 layers of transparent surfaces, the performance falls below 10 FPS.

The test scene for the table above consists of full screen layers, so each frame's linked list contains the same number of layers.

Table 1: Results with 800×600 resolution

Layer count	FPS
2	90
4	49
6	32
8	23
10	18
12	14
14	12
16	10

Table 2: Results with 1024×768 resolution

Layer count	FPS
2	51
4	27
6	18
8	13
10	10
12	8
14	6
16	5

5 Conclusions

The paper introduces a real-time method for rendering transparent, and possibly intersecting objects considering the order of the fragments. The implementation is capable of running relatively high framerates and provides a mathematically correct and more realistic result. The previous approaches were able to render simple scenes with low resolution and frame rates, thus, the method presented here demonstrates a significant advancement. In the future this technique can be used in particle systems, improving the quality of particles and eliminating artifacts when interacting with each other. Currently, we can handle about a hundred particles in real-time. The plans for the future also involve the shading of the transparent objects. The current solution uses only ambient lights and Rayleigh shading, the introduction of various light sources and shadows would significantly improve the rendering quality.

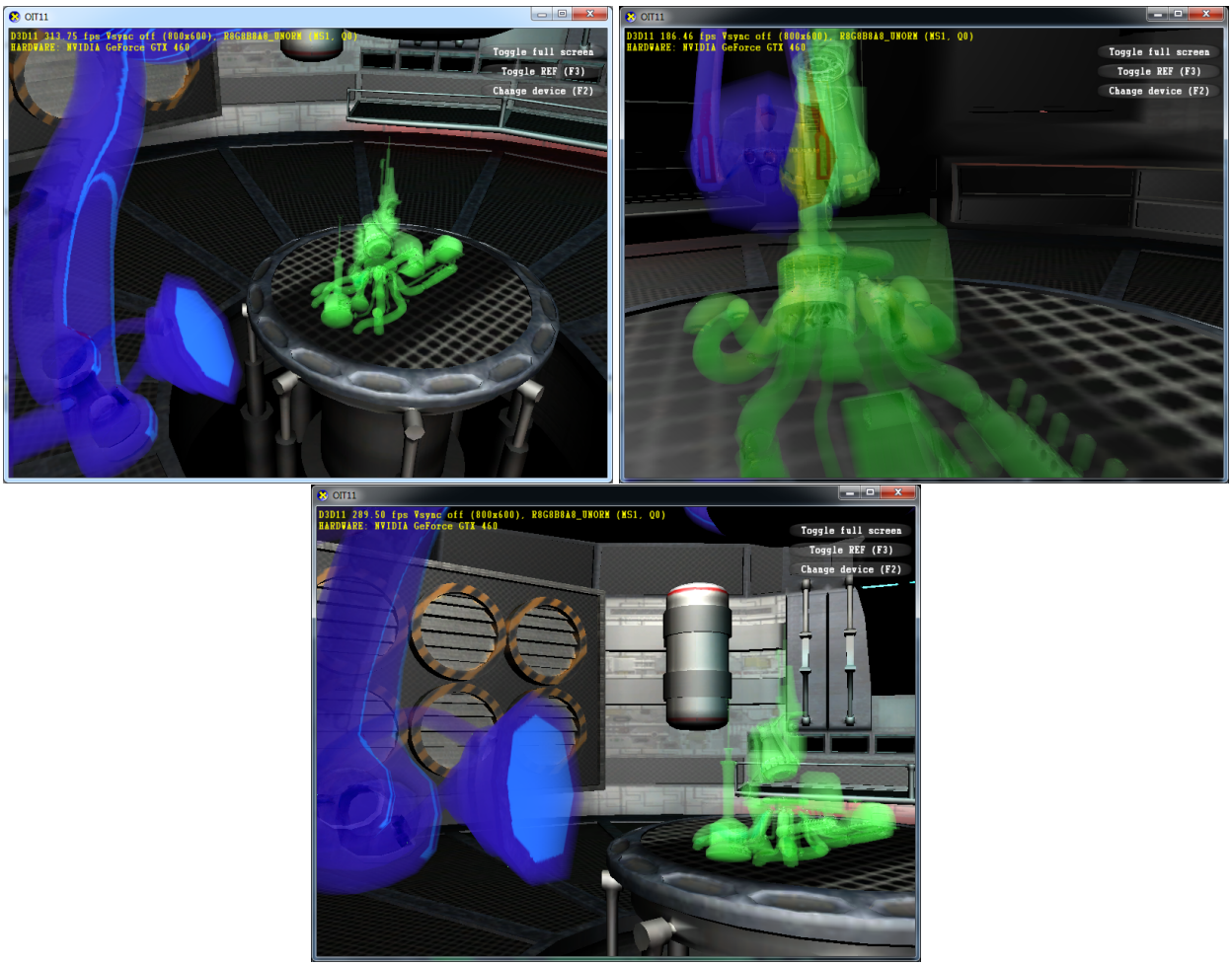


Figure 6: Rendering results (internal view of a space ship).

Acknowledgements

This work has been supported by OTKA K-719922, and by the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" (Project ID: TMOP-4.2.1/B-09/1/KMR-2010-0002). The authors are grateful to NVIDIA for donating the GeForce 480 GPU cards.

References

[1] M. de Berg. *Efficient Algorithms for Ray Shooting and Hidden Surface Removal*. PhD thesis, Rijksuniversiteit te Utrecht, The Nederlands, 1992.

[2] Oskar Elek and Petr Kmoch. Real-time spectral scattering in large-scale natural participating media. In Helwig Hauser and Reinhard Klein, editors, *Proceedings of Spring Conference on Computer Graphics 2010*, pages 83–90. Comenius University, Bratislava, 2010.

[3] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA Corporation, 2001.

[4] Nicolas Thibieroz Holger Gruen. Oit and indirect illumination using dx11 linked lists. In *Game Developers Conference*, 2010.

[5] Kevin Myers Louis Bavoil. Order Independent Transparency with Dual Depth Peeling. 2008. http://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf.

[6] L. Szirmay-Kalos and W. Purgathofer. Global ray-bundle tracing with hardware acceleration. In *Rendering Techniques '98*, pages 247–258, 1998.

[7] L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.

[8] N. Thibieroz. Robust order-independent transparency via reverse depth peeling in Direct3D 10. In Wolfgang Engel, editor, *ShaderX 6: Advanced Rendering Techniques*, pages 211–226. Charles River Media, 2008.

[9] T. Umenhoffer, L. Szirmay-Kalos, and G. Szijártó. Spherical billboards and their application to rendering explosions. In *Graphics Interface*, pages 57–64, 2006.

[10] Phil Willis. Projective alpha colour. *Computer Graphics Forum*, 25(3):557–566, 2006.

Attention & Entertainment

Saliency map augmentation with facial detection

Julia Kucerova*
Supervised by: Elena Sikudova†

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

Abstract

Visual attention is very important in human visual perception. It is the ability of a vision system to detect salient objects in an observed scene. This scientific discipline has been studied for over a century. Nowadays it is involved in the disciplines of psychophysics, cognitive neuroscience and computer science.
This paper describes several visual attention models for detecting salient objects in complex scene and focuses on a model based on local context suppression of multiple cues. Although this model is useful to capture visual attention in images containing small objects, it fails in detecting faces as salient objects.
For this reason we improved the model by adding more attention cues. We propose a method for detecting salient objects based on texture, where face detection is used as an additional attention cue.

Keywords: Visual Attention, Texture attention cue, Salient object, Face detection

1 Introduction

"Everyone knows what attention is..."

William James, 1890

Humans cannot attend to all things at once. Their visual system has the ability to pay attention to some parts of the observed scene - salient objects. Visual attention models detect these salient objects in scene.

There are two general visual processes, called $bottom-up$ and $top-down$.
The bottom-up process is task-independent. This process tries to predict which parts of the observed scene could attract more attention and computes saliency map. It could be used in machine vision, automatic detection of goals in nature scenes, intelligent image compression, etc. Salient objects in scene are for example a burning candle in a dark room or the lips and eyes of a human face, because they are the most significant elements of the face. If there are many salient objects in the scene, they become obscure because of the big amount.
The top-down process is volition-controlled and task-dependent. It drives observer's attention on one or more objects that are relevant to the observers goal when studying the scene. For example the task could be to find red car on a car park, or to count particular objects in a scene. When the observer is concentrated to find some objects in the scene, he will fob off some salient objects. For that reason some objects that are salient in bottom-up process could not be found with top-down process.
In 1967 psychologist Yarbus recorded eye movements of participants watching an image. The subjects had task to observe Repin's picture "An Unexpected Visitor" and they were asked to answer a number of different questions (Figure 1).

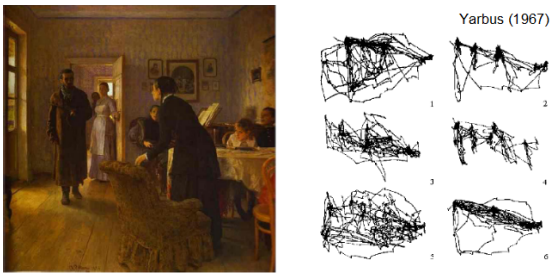


Figure 1: Repins picture was examined by subjects with different instructions; 1. Free viewing, 2. Judge their ages, 3. Guess what they had been doing before the unexpected visitors arrival, 4. Remember the clothes worn by the people, 5. Remember the position of the people and objects in the room, 6. Estimate how long the visitor had been away [1].

The motivation of our work is that visual attention is very important in human vision. We can use our knowledge about visual attention for many applications. In image compression we can compress background objects while salient objects stay untouched. Or we can use it in artefact removal algorithm to remove uninformative and distracting color boundaries [11].

Faces are very significant in human perception of the scene. This fact was studied in [5], where the authors investigated, whether faces are capable of capturing atten-

*kucerova.julia@gmail.com
†sikudova@sccg.sk

tion when compiting with other non-face objects. Their results suggest, that faces in fact attract attention.

The approach proposed in this paper is using face detection as an additional attention cue. It combines color, intensity and texture features with face detection map to get saliency map. Our work presents a model based on local context suppression of multiple cues presented by Hu [6].

This paper is organized in the following way: The work of other authors in the area of the visual attention is discussed in section 2. In section 3 the model based on local context suppression of multiple cues is described. In section 4 we describe face detection system used here and in section 5 the feature combination. In section 6 the paper presents the comparison of different methods and section 7 concludes the paper.

2 Related work

Visual attention has been studied for over a century. Early studies of visual attention were simple ocular observations. Since then the field has grown and nowadays it is involved in many scientific disciplines.

Scientists have observed human visual system, visual attention and many computational models have been proposed to predict what will attract our visual attention [2]. Human visual system is sensitive to features like changes in color, shapes, intensity etc. In some models low level features like color, intensity and orientation are used as attention cues.

Itti et al. [2] developed a visual attention model based on the behavior and the neural architecture of the early primate visual system. Authors used low level features like color, intensity and orientation as attention cues. They implemented linear center surround operation on multiscaled feature images. This images are created using Gaussian pyramids. After normalization all feature images are combined into a single saliency map. 2D winner-take-all algorithm is used for detection saliency regions in an image. Ma and Zhang [9] proposed a new approach for obtaining the saliency map. They used contrast analysis and developed a fuzzy growing technique in the visual attention model to extract salient regions. Bergum et al. proposed mathematical framework of visual attention for robotic system. In [3] they integrated object- and space-based models of visual attention.

Visual attention models have a wide use. Nowadays we find them in robotic systems, image compression, commercial industry etc. There are many approaches and systems for detecting salient objects and they are still improving.

3 Model

In this section we describe model presented by Hu [6], which is used as a base model for our approach.

Hu's model is based on local context suppression. The authors used texture as an additional attention cue for salient region detection. They also developed feature combination strategy that suppresses regions in contrast maps. This strategy uses local context information to suppress spurious attention regions and enhance the true attention regions.

Texture is very useful to capture visual attention in images containing small objects. Texture Attention Cue used in this model was obtained as follows. Image was divided into blocks, called *texture patches*. By taking the Gabor Wavelet Transform at different scales each texture patch is represented by the mean and the standard deviation. In this way mean maps and standard deviation maps were obtained. Consequently Average Mean Difference (AMD) and Average Standard Deviation Difference (ASDD) were created. Texture contrast at a patch (i, j) at any scale s and orientation k was calculated as

$$TC_{s,k}(i,j) = AMD_{s,k}(i,j) \times ASDD_{s,k}(i,j). \quad (1)$$

Consequently the final Texture contrast at patch (i, j) was calculated as

$$TC(i,j) = \sum_s \sum_k TC_{s,k}(i,j). \quad (2)$$

Local context suppression strategy for adaptive combination of multiple attention cues like intensity, color and texture is describe here. Consider an image divided into blocks, called an *Attention Patches*, each containing $p \times q$ pixels. The contrast of particular feature at a patch centered at (i, j) is calculated as

$$FV(i,j) = \frac{1}{N} \sum_{u,v} |MF(i,j) - MF(i+u, j+v)|, \quad (3)$$

where $MF(i, j)$ is the mean of the feature in patch (i, j) and N is the number of patches in its neighborhood. The contrasts at patch (i, j) for n features/attention cues are normalized to lie between $[0, 1]$. Each patch is now represented by the n dimensional feature contrast vector which is compared with other feature contrast vectors in its neighborhood and its contrast measure is suppressed if the patch and its neighbors are 'similar'. This similarity is estimated by the variance of data along eigen vectors of an $n \times n$ covariance matrix. This matrix is formed from the feature contrast vectors at a patch (i, j) and its neighborhood. The eigen values $\bar{\lambda}$ of this matrix represent the extent of similarity or dissimilarity among the attention cues. For example a large eigen value indicates large variance along the direction of its corresponding eigen vector, which implies higher discriminating power.

The suppression factor (SF) for patch (i, j) is obtained as $\tau(i,j) = \prod_{u=1}^{p} \bar{\lambda}_u$, where the $\bar{\lambda}$'s are sorted in ascending order and the parameter p controls the degree of suppression. For obtaining the saliency $S(i, j)$ for patch (i, j) the

multiple attention cues are linearly combined and the result is modulated by the SF as

$$S(i,j) = \tau(i,j) \times \sum_{u=1}^{k} FV_u(i,j). \quad (4)$$

The product of the combined map and the SF yields the final saliency map which contains the true Attention Regions. In the combined map there are spurious attention regions. Using Suppression Factor, these regions have been successfully removed [6].

4 Face detection

Detecting faces in the scene is a difficult problem. There are wide variety of faces to match, variations in lighting and shadows, presence of facial hair, possibility of scaling, angular and dimensional variances. Face detection is important in many human-computer interaction systems. There are many different approaches for detecting faces in the images: knowledge-based methods, feature invariant approaches, template matching, appearance-based methods.

In this paper we use face detection as an additional attention cue. We use two different systems for face detection and after comparison we decide for one of them. The first system is based on Rowley-Baluja-Kanade neural network. In order to better detect the faces this system was combined with skin detection. The second one is based on Viola/Jones' algorithm.

4.1 Rowley-Baluja-Kanade Face Detector

In this system we use combination of skin detection and Rowley-Baluja-Kanade (RBK) face detector. Skin color distribution used in this paper is modeled using a single 2D Gaussian distribution [12]. For face detection we used a software [10] that implements the Rowley-Baluja-Kanade *RBK* neural net face detector with some enhancements for training and recognition. Rowley's et al. face detection system is neural network-based system and authors present a straightforward procedure for aligning positive face examples for training.

As an input we have image in HSV color space. To get a faster face detection we used skin probability maps. Face detector is then applied only in the regions, where skin was detected. As an output we get regions of possible skin patches. We will label non-skin regions of the image (usually background) with 0 and possible skin regions with 1. Consequently we multiply this map with the input image. This results in set of possible face candidates that is used as input for face detection system. Consequently, after face detection, we used threshold to get binary map (Figure 2 c)).

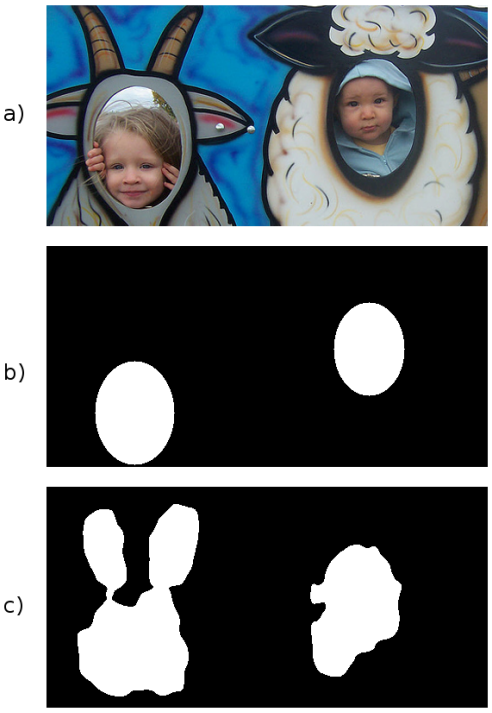


Figure 2: a) Input Image b) face detection based on 4.2 c) face detection based on 4.1

Face detector	Overlap	Left Out
Viola/Jones	80.24%	37.9%
Rowley	89.18%	79.2%

Table 1: Compare Face detectors

4.2 Viola/Jones' Face Detector

This system is used for real-time object detection. Training in this face detection system is slow, but detection is very fast. Key ideas of this face detector are integral images for fast feature evaluation, boosting for feature selection, attentional cascade for fast rejection of non-face windows.

We used the implementation of Viola/Jones' system (VJ) found in [8]. This system uses mid cumulative probability distribution point as threshold for weak classifiers.

We compared these two face detectors. As you can see in Table1-Overlap, Viola/Jones' system detected less faces as *RBK* Face Detector because of frontal detection. However VJ system has much less false positive detections than *RBK* Face Detector (Table1-Left Out). In our system we need good face detection with the least possible false positive detections. For that reason we decided to use VJ system.

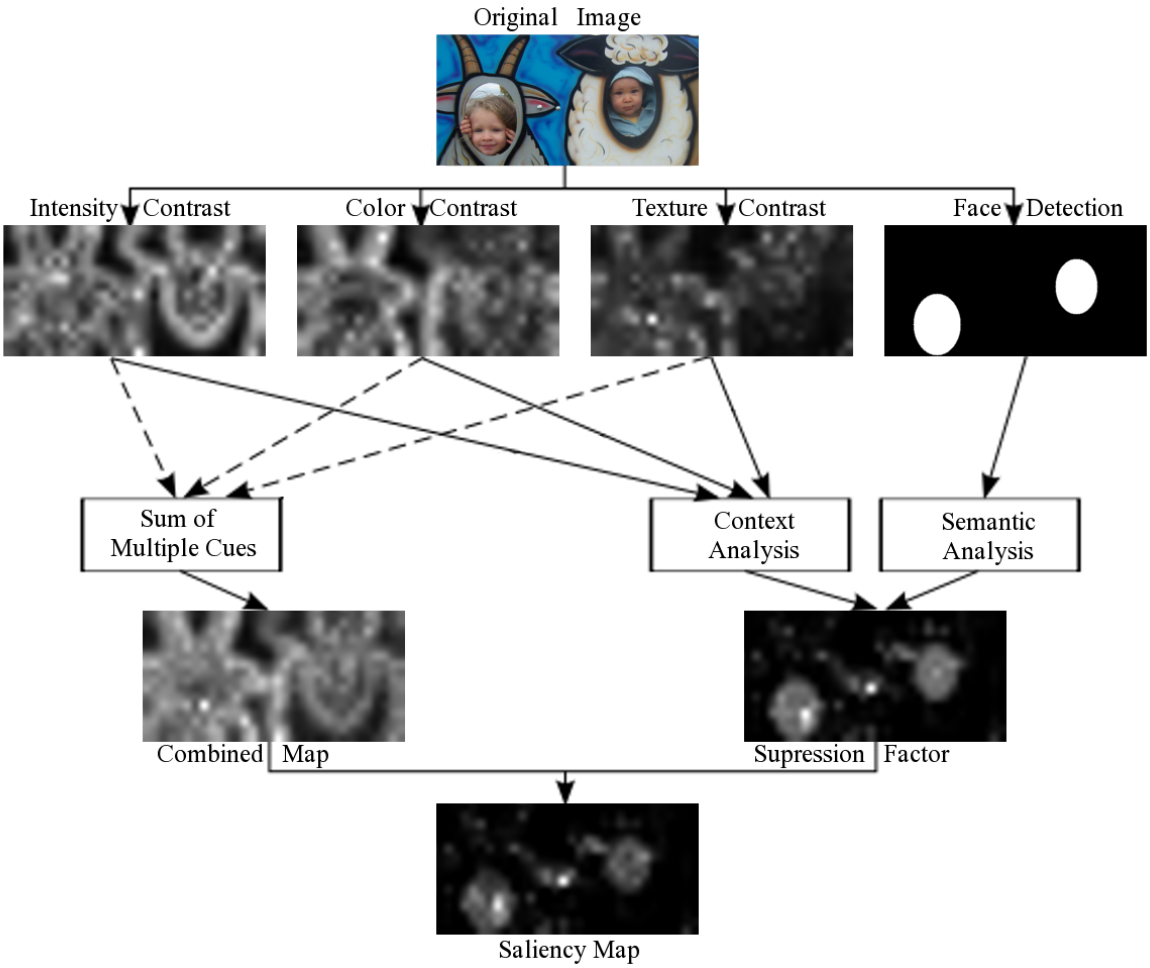


Figure 3: Features combination

5 Features combination

The combination of features that yields the final saliency map that includes only the true attention regions is a hard problem. Some approaches suggesting linear combination [2], other suggest some post-process, weighted combination etc.

In this paper we used and modified feature combination proposed in [6]. As shown in Figure 3, we have four features: color, intensity, texture and face detection maps.
Contrast maps for intensity, color and texture are obtained the same way as in [6] and face detection map is obtained by Viola/Jones' Face Detector.

As a first step of feature combination we sum together and normalize three contrast maps (color, intensity, texture) to get the Combined map.
We derive the suppression factor by building up the suppression map from color, intensity and texture as proposed in [6]. We combine this map with the map for face detection to get the suppression factor, which highlights significant regions as well as faces.

Suppression factor is a map consisting of darker regions representing high suppression factor and brighter regions representing low suppression factor. That means, that brighter regions are more significant than darker regions.
Consequently we multiply this map with Combined map. With this process we get final Saliency map for input image.

6 Results

This section summarizes the results of the proposed approach.

We used images from Visual Object Classes database [4] for testing, which is a benchmark in visual object category recognition and detection. It contains standard dataset of images and annotation, and standard evaluation procedures and significant variability on terms of object size, orientation, illumination etc. In this dataset images are sorted in many different classes as persons, animals, indoor images, vehicles etc.

We compare our system with the system based on Itti et

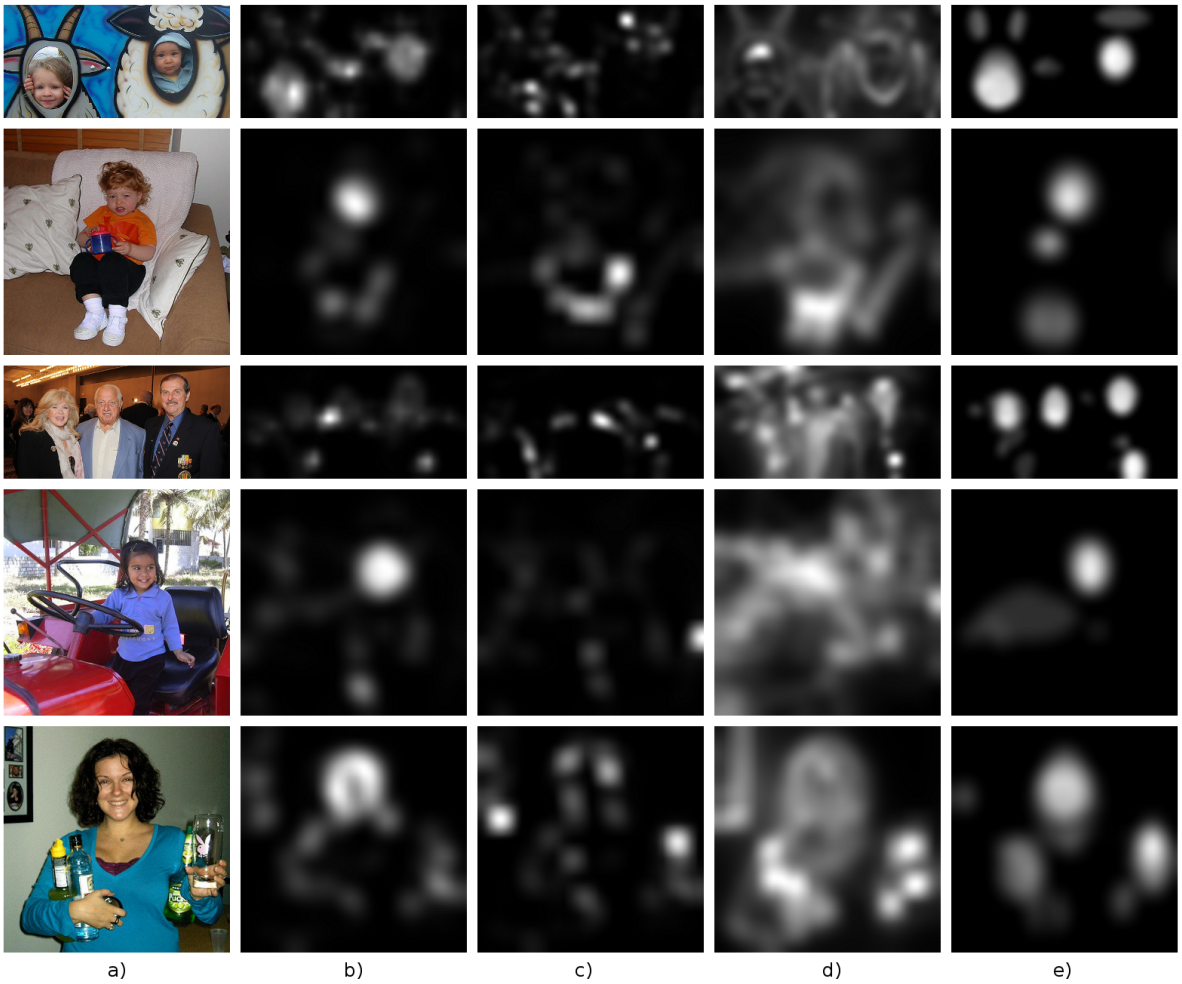


Figure 4: Experiment results a) Original Image; b) Saliency map using proposed method c) Hu‘s model [6] d) Itti‘s model [7] e) Manual combination

al. [7] and model proposed by Hu et al. [6]. For comparison we used salient regions obtained by manual inspection of the images. We asked several(11) observers to highlight significant regions. These maps were than summed together, normalized and then thresholded. In the next phase of our work, we will use data from eye tracking system to obtain real saliency data and compare them with our results.

For comparison our results we used symetric Kullback-Leibler divergence

$$KLD(P,Q) = \sum_i (P(i) - Q(i)) * \log \frac{P(i)}{Q(i)}, \qquad (5)$$

where P is saliency map obtained by Itti‘s model [2], Hu‘s model [6] or our model, and Q is the manual map. When the two probability densities are identical, KLD is null. The lower KLD, the better model.

As you can see in Table 2 our approach has the lowest KLD. Salient regions detected using Itti‘s model contain faces, but cover a significant portion of the input image.

Visual Attention Model	KLD
Proposed method	1.1377
HU	2.2807
ITTI	1.6642

Table 2: Compare Visual Attention Models

Hu‘s model is very useful in images obtaining for example texture foreground in non-texture background, but as you can see, it is less successful in images containing faces and bigger objects. Our approach detects salient regions of various sizes as well as faces.

Noticeable, that regions, which contain faces are more salient than other parts in the image.
Although the true attention regions are very subjective for each observer, they could be detected to a large extent. By using our model we can detect salient objects more accurately.

7 Conclusions and Future work

As a conclusion the best results achieved using this model are comparable with other visual attention models as Itti's model [2], Hu's model [6]. Our approach is based on the idea, that faces take more attention in the observed scene. We adapt input model [6] by adding face detection as an additional attention cue.

Data obtained with this approach are very useful. Detection of saliency regions in the observed scene is being used in image compression. Compression using visual attention rests in fact, that salient regions could be less compressed than non-salient regions.

In the next phase of our work the face detection process will be improved by exploring different feature combination and/or more color spaces. We will also improve the suppression factor for better results.

As a future work we planed to use eye tracking of subjects to obtain real saliency data and compare them with the proposed method.

8 Acknowledgments

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

References

[1] K. Cater, A. Chalmers, and G. Ward. Exploiting visual tasks for selective rendering. In *Eurographics Symposium on Rendering*, pages 270–280. Eurographics, 2003.

[2] L. Itti et al. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11).

[3] M. Begum et al. Object- and space-based visual attention: An integrated framework for autonomous robots. pages 301–306. IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2008.

[4] M. Everingham et al. Visual object classes database. [Online] `http://www.pascallin.ecs.soton.ac.uk/challenges/VOC/`, 2006-2009.

[5] S.R.H. Langton et al. Attention capture by faces. *Cognition*, 107:330–342, 2008.

[6] Y. Hu et al. Adaptive local context suppression of multiple cues for salient visual attention detection. In *IEEE International Conference on Multimedia and Expo*, pages 1–4, 2005.

[7] J. Harel. A saliency implementation in matlab. [Online] `http://www.klab.caltech.edu/~harel/share/gbvs.php`, 2010.

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

[9] Y. F. Ma and H. J. Zhang. Contrast-based image attention analysis by using fuzzy growing. In *ACM International Conference on Multimedia*, pages 374–381, 2003.

[10] S. Sanner. Rowley-baluja-kanade face detector. [Online] `http://users.cecs.anu.edu.au/~ssanner/Software/Vision/Project.html`, 2005.

[11] F. Stentiford. A visual attention estimator applied to image subject enhancement and colour and grey level compression. In *International conference on Pattern Recognition (ICPR(3))*, pages 638–641. IEEE, 2004.

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

Do-It-Yourself Eye Tracker: Impact of the Viewing Angle on the Eye Tracking Accuracy

Michal Kowalik*
Supervised by: Radoslaw Mantiuk†

Faculty of Computer Science
West Pomeranian University of Technology in Szczecin
Poland

Abstract

In the paper we research the relations between the eye tracker accuracy and the human view angle. We measure the accuracy of the gaze point estimation for a typical and wide view angles and discuss limits of the field of view covered by an eye tracker. The measurements are captured during perceptual experiments with human observers. We built eye tracking station consists of our own construction eye tracking glasses and ITU Gaze Tracker software. It's based on the pupil-detection technique. We used this eye tracker station, called Do-It-Yourself, in the experimental hardware setup. We conduct perceptual experiments to measure eye tracker accuracy for increasing view angles.

Keywords: eye tracking,eye tracker hardware,view angle estimation,subjective experiments

1 Introduction

Eye tracking devices determine the position of the eye in space and compute position of a gaze point and a gaze direction. This information is utilised in science and technology, e.g. to test peoples' preferences concerning advertisement, or to control computer via the eye tracker interface, etc.

The progress in technology increases availability of computer monitors with large diagonals. They cover wider viewing angle and strengthen impression of the visualisation realism. Most probably, we can expect a display that covers the whole 180° degrees of human visual angle in the near future. The eye tracking technology must be adjusted to these parameters. However, other limitations of Human Visual System (HVS), like foveal vision, also influence the eye tracker operation.

The main objective of the article is determine a relationship between eye tracking accuracy and visual angle. We measure accuracy of eye tracker for small and large view angles. In research we used Do-It-Yourself (DIY) eye tracker: our own construction eye tracking glasses in cooperation with ITU Gaze Tracker software. We built this low cost eye tracker to gain full control over the eye tracking pipeline. We conduct perceptual experiments to measure eye tracker accuracy for increasing view angles. We determine the limits of accuracy resulting from eye tracker hardware design and possibilities of gaze estimation algorithms.

Section 2 presents basic terminology and classification of eye tracking techniques. Design of DIY eye tracker is depicted in Section 3. Section 4 contain the eye tracker accuracy concept. Section 5 described experimental procedure together with the discussion of results. We conclude the paper in Section 6.

2 Background and previous works

The tracking of viewing direction have been known in science for many years. With eye tracking techniques we are able to identify the place which a user is looking at. This discipline deals with the measurement, recording and analysing data about the location and movements of the eyeballs. The results of the eye tracker work is the point of regard. A subset of the points of regard is known as an region of interest (ROI). Science knows eye tracking analog methods for example contact lenses [13] or electro-oculogram [1]. These methods are invasive and come into a strong interaction with the user. Modern eye tracking systems use the image of eye obtained by video equipment to calculate the point of regard. They are more comfortable for users than the intrusive methods. We distinguish two types of video based eye-tracking systems [6, 5]: mobile - the camera is mounted on the head and remote - the camera is located near the monitor. The mobile system consists of glasses or a helmet with mounted cameras that record the movements of an eye or eyes.Remote system consists of a camera located close to the monitor in the front of observer.

Eye tracking systems work in visible light [10] or in infrared light, based on the image of one eye [15] or stereoscopic vision [4].The infra-red eye apparition allows to locate the dark pupil, the bright pupil and the corneal reflection (See Fig. 1). [6]

*kkowalik@wi.zut.edu.pl
†rmantiuk@wi.zut.edu.pl

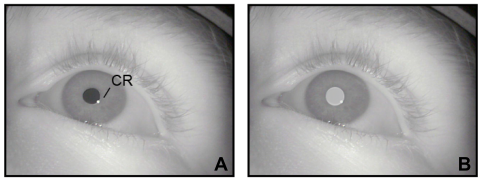


Figure 1: Appearance of human eye in infrared light: A) dark pupil and corneal reflection - a flash located on the surface of the eye, B) bright pupil.

The changes analysis of the vector connecting the centre of the pupil and corneal reflection is a classic example of the remote eye tracking method. Assuming that the eye is a sphere and rotates around its own centre and camera with infrared source is stationary then a corneal reflection position is unchanged to different gaze direction. The Corneal reflection can be used as a reference point. The centre of the pupil (or iris) with a corneal reflection create a vector which is mapped to the coordinates of the screen during the calibration process. This solution is non-invasive and allows user for small head movements. [12]. The remote methods divide into: based on changes of the pupil - eye corner vector [19], mapping of four corneal reflections [17, 18, 7] and based on the three-dimensional model of the eye [15].

Tracking pupil centre is a method which is used in mobile eye trackers (mounted on the head). It uses dark pupil, thresholding and model fitting method. The position of pupil centre is compensated with parameters derived in the calibration process. The result is an estimated point of gaze [11, 16]. The algorithms which works in the visible light use the centre of the iris to calculate point of regard.

The mobile eye tracking systems are less comfortable for user than the remote systems because some device must be wear. However, they range is not limited to display screen space and they can operate e.g. in the real environment.

3 Do-it-yourself Eye Tracker

In our project was created Do-it-yourself Eye Tracker station (DIY ET). The main goal of the project was to create inexpensive and simply in construction eye tracking tool. DIY ET base on self constructed eye gaze tracking glasses supported by open source eye tracking application.

3.1 Eye tracking glasses

DIY ET belongs to the group of head mounted eye trackers. It works in infra-red spectrum using dark pupil effects. The point of gaze is calculated by the position of pupil centre.

DIY ET consists of two main parts: eye tracking glasses and computer with ITU Gaze Tracker software. The construction of the eye gaze tracking glasses was based on articles [14, 3, 9].The glasses are made of off-the-shelf component. The main part of glasses is the capture module (Fig. 3D). It is responsible for providing an image of the eye to the computer. This module was created by using the Microsoft LifeCam VX-1000. We mounted a suitable filter in camera lens that allows capturing images in infrared light (Fig. 3A-B). The glasses are connected to a computer via USB port. Based on the USB technical specification a infra-red illumination system was integrated with the capture module. The infra-red LEDs are located on the capture module and supplied by USB cable (Fig. 3C). This solution is very practical. The capture module was placed at the end of the aluminium wire and then mounted to the modified safety glasses frame (Fig. 4).

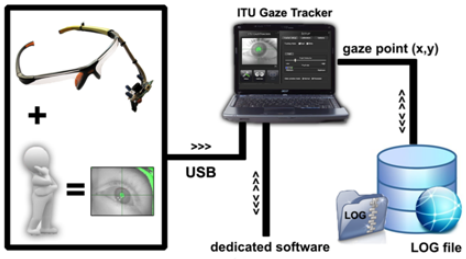


Figure 2: DIY Eye Tracking setup.

The created glasses provide a picture of an eye to the computer by USB. Then supported application computes the point of gaze and returns in the form of coordinates (X,Y). The coordinates are stored in LOG file or transferred directly to another application via client-server.

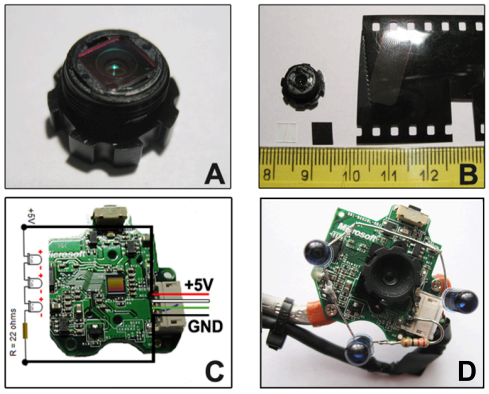


Figure 3: The creation of capture module: A) original lens from Microsoft VX-1000 web cam with visible light filter, B) preparation of IR filter, C) LEDs wiring diagram, D) capture module of eye tracking glasses.

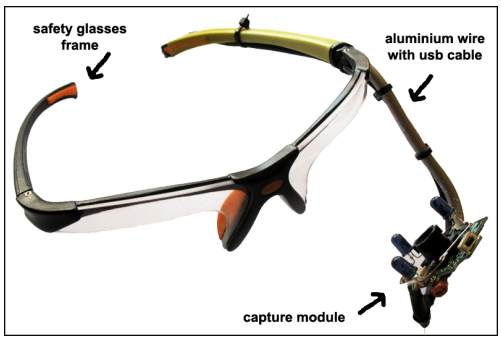


Figure 4: DIY eye tracking glasses.

3.2 ITU Gaze Tracker

The DIY eye tracker is controlled by the ITU Gaze Tracker software. ITU Gaze Tracker [2] is application designed in IT University of Copenhagen with open source licence [1]. The application estimates the gaze point by mapping the centre of the pupil to screen coordinates using the parameters obtained in the calibration process. Image of eye in infrared light is captured in consecutive frames. The pupil centre is determined and its movements are being tracked.

4 Evaluation of eye tracker accuracy

We discuss accuracy of eye tracking systems. The eye tracker accuracy is measured in degrees of visual angle.

4.1 Human field of view

The whole human viewing angle is about 180° horizontally and 130° vertically. However, the binocular field of vision covers only about 120° horizontally. Additionally, The details can be read only by fovea - a part of the eye located in the middle of the macula on the retina. Fovea extends from 1° to 5° the human view angle.

An eye tracker should operate in a view field that do not force head movements. One assumes that it is not more than 120° of binocular vision. For observer sitting in 50 cm distance from a screen, a display should be up to 170 cm wide.

4.2 Gaze angle

During calibration an observer is asked to look at a set of target points displayed in different position on the screen. The image of the eye is recorded and the eye pupil centre location is calculated. Correlation between calculated position of the pupil centre and known position of the target points is used to approximate coefficient a_{0-5} and b_{0-5} of the polynomial:

$$\begin{cases} screen_x = a_0 + a_1x + a_2y + a_3xy + a_4x^2 + a_5y^2 \\ screen_y = b_0 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2, \end{cases} \quad (1)$$

where $(screen_x, screen_y)$ are the gaze point coordinates on the screen, (x,y) are the coordinates of the centre of the pupil [2]. The accuracy of calibration process significantly affects error arising during eye tracker operation. This accuracy of eye tracker is determined by indicating the differences in position between the reference points with known position and measure gaze points. The accuracy is expressed in degrees of visual angle.

4.3 Error factors affecting accuracy of eye tracker

A significant error affecting the accuracy of the gaze point estimation is the head movement. We use a chin-rest to stabilise the head and increase the DIY eye tracker accuracy. Other solutions utilise algorithms that compensate head movements [8] or use additional the head trackers.

For the wide view angles the eye tracker cannot detect pupil centre accurately. The extreme situation is presented in Figure 5B where pupil was not detected by image processing software.

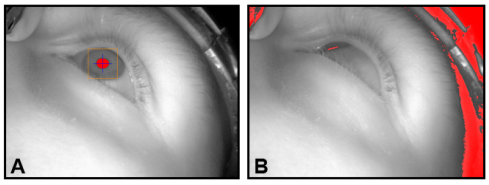


Figure 5: Detection of the pupil for standard (A) and wide view angle (B).

The DIY eye tracker is equipped in one camera and takes image of only one eye. The measurement error cannot be compensated by the data from the second eye. Another sources of errors encompass inaccuracies of the pupil centre extraction, variation of lighting and of shadows covering the eye. Changing illumination cause confusion in getting clear image of eye. For this reason the software cannot measure position of pupil centre. It is really important to provide stable lighting during research.

Results of DIY ET have a high standard deviation. It is characteristic for the data from the eye tracker and arises from the physiology vision of the human eye. Human eye fix independent of human will around gaze point. It can creates a outliers. Filtering the outliers may be the solution to reducing standard deviation.

5 Perception study

The goal of the tests was to find relation between the accuracy of eye tracker and an observer view angle. During tests we used hardware setup based on the DIY eye tracker.

[1] http://www.gazegroup.org/

5.1 Hardware setup

Our experimental setup is presented in Figure 6. It consists of the DIY eye tracker controlled by the ITU Gaze Tracker software (2.0 of this software). The application was activated on PC equipped with Windows XP SP3 operating system, AMD Athlon 64 X2 Dual Core Processor 3600+, NVIDIA GeForce 7600 GS 512MB graphics card and 3GB DDR2 RAM. The target points were displayed on Samsung SyncMaster 2233sn with the screen dimensions 46.5 x 27 cm, and native resolution 1920x1080 pixels (60Hz).

We used the chin-rest adopted from the ophthalmic slit lamp. The tests were conducted for three distances from eyes to the screen: 70 cm, 50 cm and 30 cm. Reduction of distance corresponds to increase an angle of view.

We used our own construction eye tracking glasses (Fig. 4). The glasses worked with 640x480 pixels resolution and 30 fps frequency.

Figure 6: Hardware setup used during experiments.

5.2 Participants

Ten users with an age from 21 to 56 participated in our experiment (two woman's and eight men). Seven participants had normal vision, three of them had corrected vision with lens. We asked each participant to repeated the experiment three times for each distance. In all we have ninety measurements, thirty for each distance. The whole experiment for one person lasted less than 8 minutes. Participants were aware that accuracy of the eye tracker is tested, however they do not know details of the experiment.

5.3 Procedure

The participants were asked to wear the DIY eye tracker and use the chin-rest to stabilise the head. They looked at the target points that were displayed on the screen one by one in random order as white circles. The procedure was repeated for 70 cm, 50 cm and 30 cm distances by tuning position of the monitor. The monitor and participant were located on the same axis of symmetry.

5.4 Results

Figure 8 presents all data collected during the test procedure. Location of the target points is marked by red circle, the observers' gaze points are depicted as a blue dots. Distribution of gaze points for shortest distance (highest view angles) is more spread out and does not follow the target point position very well. It results higher accuracy error. Figure 7 presents box plots of average error for sixteen target points. The central mark (red line) indicates median value of the error, the edges of the box are the $25th$ and $75th$ percentiles, the whiskers extend to the most extreme data points not considered outliers. Outliers are plotted individually as red crosses. The blue horizontal line indicates view angle error equal to one degree of visual angle.

For $70cm$ distance from screen, average error for all target points amounts to $0.34deg$ (with standard deviation equal to $0.49deg$) for horizontal direction and $0.45deg$ (with standard deviation equal to $0.58deg$) for vertical direction. For higher view angles ($50cm$ distance), the errors increase to $0.59deg$ (standard deviation $0.75deg$) and $0.55deg$ (standard deviation $0.59deg$) for horizontal and vertical direction respectively. For highest view angles ($30cm$ distance), the errors increase to $1.77deg$ (standard deviation $1.71deg$) horizontally and $1.20deg$ (standard deviation $1.03deg$) vertically.

The results of our experiment demonstrated high influence of observers' viewing angle on eye tracker accuracy. The best accuracy was measured for largest distance from screen (70 cm). Precision dropped with reduce the distance and is worst for 30 cm (widest angle of view). However, we did not notice this relationship for the individual target points.

There are no regular fluctuations of the error for central and extreme target points. In Fig. 8-bottom we observed dependence between four centre points and rest outside points. The gaze points for the centre reference points have got worse accuracy than the outside gaze points. They are strong shifted toward outside. It is not expected dependence because these four centre points are placed in sharp field of view (for 30 cm distance) contrast to the outside points. Large group of outside points (12 points) affect to calculation of polynomial terms stronger than group of centre points (4 points). In this case the outside points have got better accuracy than the centre gaze points. This error is consequent of calibration method implemented in the ITU Gaze Tracker application.

6 Conclusions and future work

The main contribution of this paper is indication relation between accuracy of eye tracking system and viewing angle. Precision drops with wider angles of view. Accuracy of the DIY eye tracker is close to 0.6 degree for standard viewing angles (up to 30 degrees, 50 cm distance from screen). It is satisfactory result considering low cost of the

eye tracker. However, the accuracy decreases to 1.7 degrees for wide view angles what seems to be unacceptable in most applications.

In future work we plan to implement validation of DIY eye tracker which allows to achieve more accurate results and shows better the relationship between the eye tracker and the wide angle. Combination of eye tracking with head tracking seems to be the solution of head movements problem. We plan to build low cost head tracking device and integrate it with the DIY eye tracker.

References

[1] Kaufman A., Bandopadhay A., and Shaviv B. An eye tracking computer user interface. *Proc. of the Research Frontier in Virtual Reality Workshop, IEEE Computer Society Press*, pages 78–84, 1993.

[2] Javier San Agustin, Henrik Skovsgaarda, and Dan Witzner Hansen John Paulin Hansen. Low-cost gaze interaction: Ready to deliver the promises. *CHI*, 2009. Boston, Massachusetts, USA.

[3] J. Babcock, J. Pelz, and J. Peak. The wearable eyetracker: A tool for the study of high-level visual tasks. *Proceedings of the Military Sensing Symposia Specialty Group on Camouflage*, February 2003.

[4] Yongqin Cui and Jan M. Hondzinski. Gaze tracking accuracy in humans: Two eyes are better than one. *Neuroscience Letters*, 396:257–262, 2006.

[5] A.T. Duchowski. *Eye Tracking Methodology: Theory and Practice (2nd edition*. Springer, London, 2007.

[6] Riad I. Hammoud. *Passive Eye Monitoring - Algorithms, Applications and Experiments*. Springer-Verlag Berlin Heidelberg, 2008.

[7] You Jin Ko, Eui Chul Lee, and Kang Ryoung Park. A robust gaze detection method by compensating for facial movements based on corneal specularities. *Pattern Recognition Letters*, (29):1474–1485, 2008.

[8] Susan M. Kolakowski and Jeff B. Pelz. Compensating for eye tracker camera movement. *Proceedings of the 2006 symposium on Eye tracking research and applications*, pages 79–85, 2006. California.

[9] D. Li, J. Babcock, and D.J. Parkhurst. openeyes: A low-cost head-mounted eye-tracking solution. *Proceedings of the ACM Eye Tracking Research and Applications Symposium*, 2006.

[10] Dongheng Li and Derrick Parkhurst. Open-source software for real-time visible-spectrum eye tracking. *Human Computer Interaction Program, Iowa State University, USA*, 2006.

[11] Dongheng Li, David Winfield, and Derrick J. Parkhurst. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. *Proceedings of the IEEE Vision for Human-Computer Interaction Workshop at CVPR*, 2005.

[12] C.H. Morimoto and M. Mimica. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98:4–24, 2005.

[13] D.A. Robinson. A method of measuring eye movements using a scleral search coil in a magnetic field. *IEEE Trans. Biomed. Eng.*, 10, 1963.

[14] Jason S.Babcock and Jeff B. Pelz. Building a lightweight eyetracking headgear. *Eye Tracking Research & Application*, 2004.

[15] Jian-Gang Wanga, Eric Sungb, and Ronda Venkateswarlua. Estimating the eye gaze from one eye. *Computer Vision and Image Understanding*, 98:83–103, 2005.

[16] David Winfield. Constructing a low-cost mobile eye tracker. 2005.

[17] D. Yoo, J. Kim, B. Lee, and M. Chung. Non contact eye gaze tracking system by mapping of corneal reflections. *Proc. of the Internat. Conf. on Automatic Face and Gesture Recognition*, pages 94–99, 2002.

[18] Dong Hyun Yoo, Myung Jin Chung, Dan Byung Ju, and In Ho Choi. Non-intrusive eye gaze estimation using a projective invariant under head movement. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation Orlando, Florid*, May 2006. Florida.

[19] J. Zhu and J. Yang. Subpixel eye gaze tracking. *Proc. of the 5th IEEE International Conference on Automatic Face and Gesture Recognition*, pages 131–136, 2002.

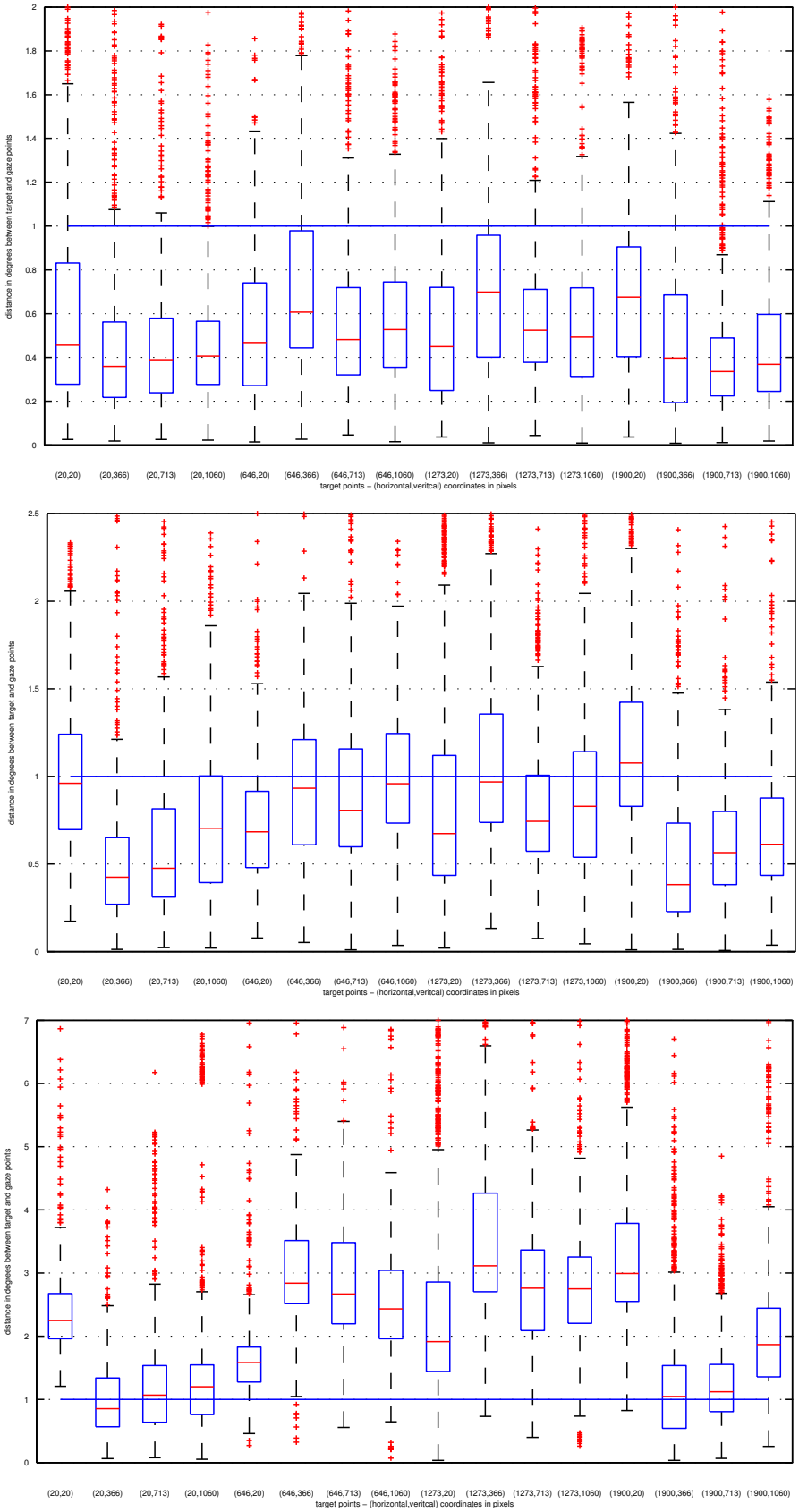


Figure 7: Average distance between target and gaze points in degrees of view angle for 16 target directions. Observers' eyes located 70 cm (top), 50 cm (middle) and 30 cm (bottom) from the screen.

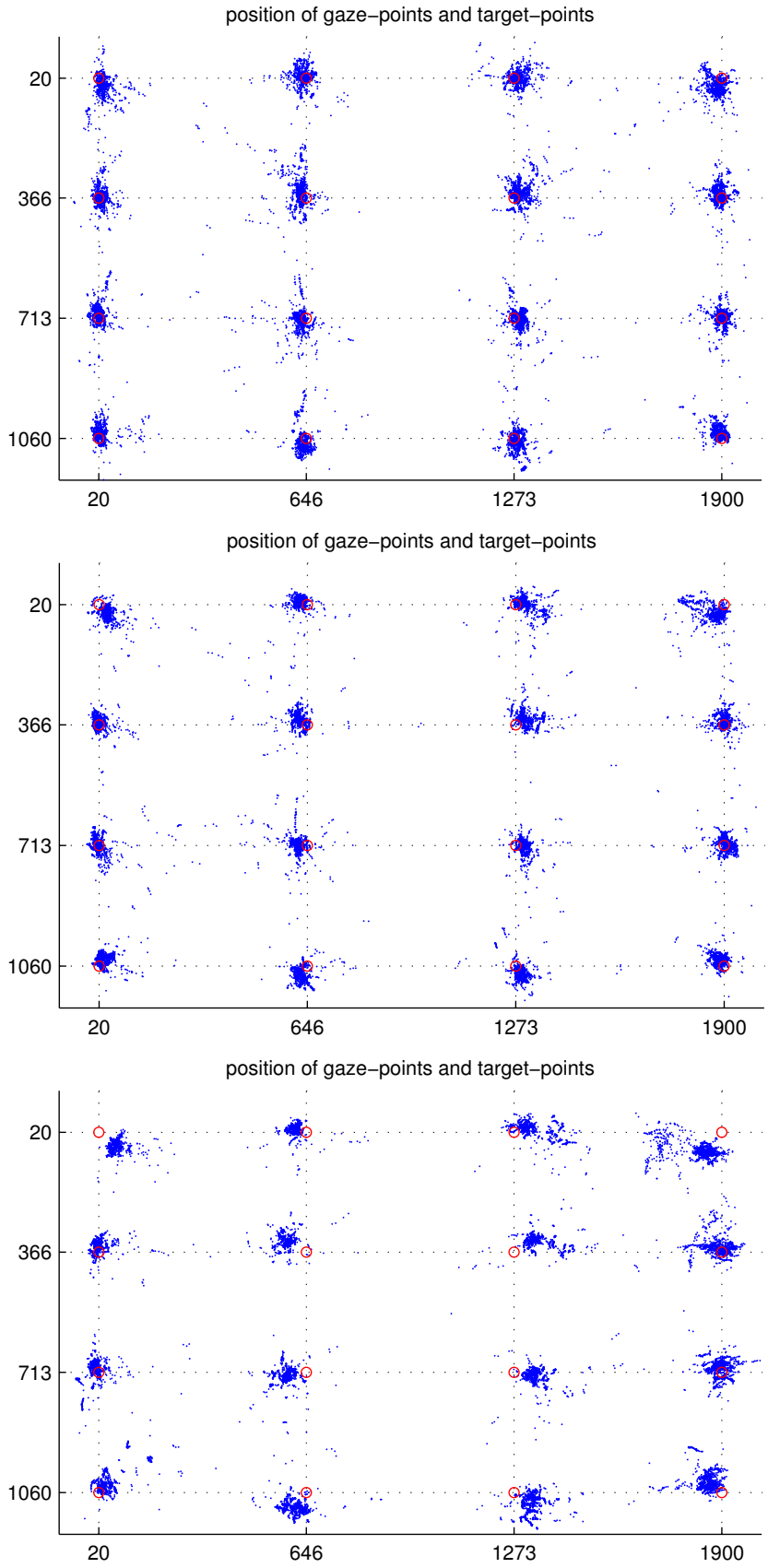


Figure 8: Position of the target-points (red circles) and measured gaze-points (blue dots). Observers' eyes located 70 cm (top), 50 cm (middle) and 30 cm (bottom) from the screen. Point positions in pixels.

Content Creation for a 3D Game with Maya and Unity 3D

Matthias Labschütz[*] Katharina Krösl[†]
In alphabetical order: Mariebeth Aquino[‡] Florian Grashäftl[§] Stephanie Kohl[¶]
Supervised by: Reinhold Preiner

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

Abstract

'Dynamite Pete'[1] is a 3D game we developed with Autodesk Maya and Unity 3D in a team of 26 computer science students with varying skills and expertise in content creation. A game development pipeline explaining the production of the game from concept to release is presented. In addition, this paper explores the challenges faced by the project staff and outlines the experiences gained during the project's implementation.

Keywords: content creation, content creation pipeline, game production pipeline, Autodesk Maya, Unity 3D

1 Introduction

Nowadays, the quality of graphics and realism of games is constantly increasing, since consumers are always demanding a more realistic look and feel in their games. This means that improvement of renderings, outstanding content, more believable animations and more authentic behavior of artificial intelligence are needed. Therefore the work of artists and animators is crucial for the success of a game and the prosperity of a game development studio.

In 2010 a group of 26 students attending 'Maya-Course 2' at the Vienna University of Technology performed a game production process in a game development studio. The ultimate goal of this exercise was to produce a video game. Unity 3.1 [10] was chosen as game engine, but the focus was placed on 3D content creation for real time application, using, besides other tools, Autodesk Maya 2011. The game development team brought in different sets of skills which ranged from beginners with hardly any knowledge about content creation to students with particular profession. Every member of the team was given at least one role and had to fulfill tasks corresponding to their job title. The assigned positions covered Project Management, Technical Direction, Art Direction and different Artists. The main areas of these artists were Modeling and Sculpting, UV-Layout, Texturing, Lighting, Rendering, Rigging, Animation, Level-design and Sound. The resulting game 'Dynamite Pete' is a comic-style Western-Adventure where the player plays the role of the antihero named Pete and has to escape from a canyon. Figure 1 shows an example screenshot of the finished game.

Figure 1: In-game screenshot of the game 'Dynamite Pete'.

This paper explains a basic strategy for the professional development of a content-intensive video game in a large team and shares experiences from our project. In the following, we will shortly introduce the tools that were used (Section 2), schematize the workflow of our game development process (Section 3) and finally explain some selected challenges that had to be overcome during the creation of 'Dynamite Pete' (Section 4).

2 Tools

The main tools used throughout the project were Maya for modeling, animating and rendering and the Unity game

[*]Technical Director Workflow: e8971103@student.tuwien.ac.at
[†]Artist: e0325089@student.tuwien.ac.at
[‡]Producer: e0326746@student.tuwien.ac.at
[§]Art Director: e0300310@student.tuwien.ac.at
[¶]Technical Director Unity: e0626088@student.tuwien.ac.at
[1]http://www.cg.tuwien.ac.at/maya/

engine for implementation. In addition, image, audio and video editing tools as well as drawing, sculpting and communication tools were used.

2.1 3D Animation and Modeling Software (Autodesk Maya)

Autodesk Maya (freely available for educational purposes[2]) was chosen as 3D modeling, animation and rendering tool in this project. Maya provides artists with an end-to-end creative workflow [4]. As a professional tool it is very complex and offers a great number of features. Despite a steep learning curve, using Maya in its whole complexity requires a long-winded learning process. Another drawback is the lack of forward compatibility, which means all the artists had to work with the same version of Maya irrespective of their preferences.

2.2 Game Engine (Unity 3D)

In this project, the free version of the game engine Unity 3.1 was chosen for the production of 'Dynamite Pete'. There are different export options in this game engine, each one dedicated to another platform (e.g.: Web Player, PC and Mac Standalone, iOs, Android, Xbox 360, PS3), which eases the development for different consoles or devices. Unity attracts especially small and middle-sized development studios who hardly invest in expensive high-end rendering engines. Furthermore prototyping and game development is very quick due to the WYSIWYG ("what you see is what you get") editor which allows instant changes and live editing. In addition, Unity provides multiple built-in shaders and effects as well as a physics engine and collision detection.

For a students' project the most important reason for choosing Unity is the fact that it is very easy to use and to learn. Developing with Unity is mainly based on drag and drop with occasional adapting of scripts rather than writing code. Apart from shaders and effects which can simply be turned on in some game settings, Unity provides numerous scripts which can be dragged onto 3D models. These scripts act for example as character controllers, follow up cameras or other important features. However, Unity lacks of integrated modeling abilities, which is the reason for using Maya as external modeling tool.

Another major drawback of the free version of Unity is its lack of SVN support. This makes it difficult for multiple programmers to work concurrently on one single project, which will be discussed in detail in section 4. Like Autodesk Maya, Unity is not forward compatible, but overall the advantages outweigh the disadvantages. However, the use of Unity Pro is advisable, due the previously mentioned restrictions in the free version.

[2] http://usa.autodesk.com/maya/trial/

3 General Game Production Pipeline

A game production pipeline is basically a concept of workflow management for use in the game development process. The phases of this pipeline are certain tasks that need to be fulfilled until the release of a video game.

Figure 2 gives a detailed overview of the development process of this project, as it will be explained in the following chapters, showing the main five tasks in color.

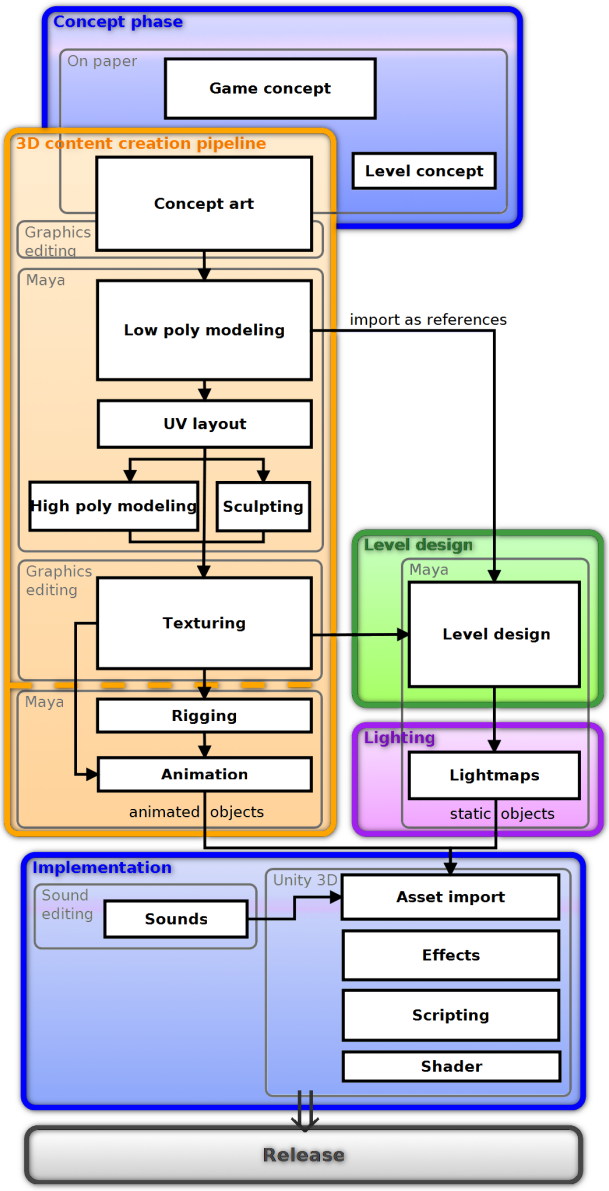


Figure 2: Game production pipeline overview: Concept, content creation pipeline, level design, lighting and implementation as organized in this particular project.

Some of these tasks require to be carried out sequentially as presented in the following. Nevertheless, to increase productivity, it should be a goal to break up this sequential approach, to allow people to work in parallel on different tasks.

Since this very project was an attempt to implement the workflow in a game development studio at a larger scale with very limited production time, it was necessary to follow a pipeline approach without iterations. The scope of this project was 3D modeling, with focus on content creation. Other aspects, like game-design, were of minor interest. The following paragraphs describe the production stages of the project in detail.

3.1 Concept Phase

During the concept phase a small team drew an outline of the plot, the setting and the game mechanics. The game's environment was chosen to be a wild-west desert. A comic style was preferred over a photorealistic style and the goal of the game was set to collecting dynamite for breaking out of a hostile canyon.

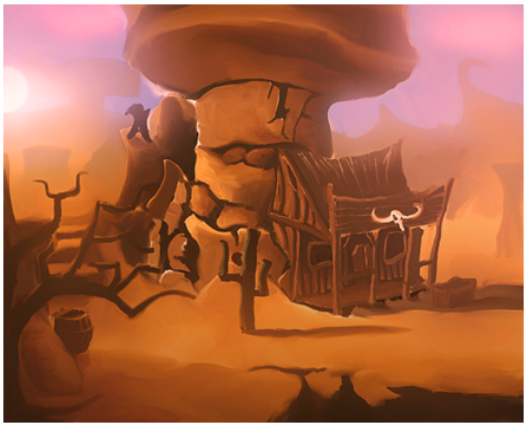


Figure 3: Early painted concept art

After this stage, the look and feel of the game content had to be worked out, requiring multiple revisions of concept arts. An example is shown in Figure 3. Inspiration was coming from all different sorts of media, starting from similar computer games and movies to comics and music. Color and proportions play an important part in the game's visual style. A dirty textured comic look was chosen, inspired by 'Star Wars - the Clone Wars Series' [9] and Woody, the Cowboy in 'Disney Pixar's Toy Story' [7].

3.2 3D Content Creation Pipeline

The orange box in Figure 2 illustrates the 3D concept creation pipeline which contains the stages every 3D model has to pass from concept art to the final model. The following paragraphs describe these stages in detail.

3.2.1 Concept Art

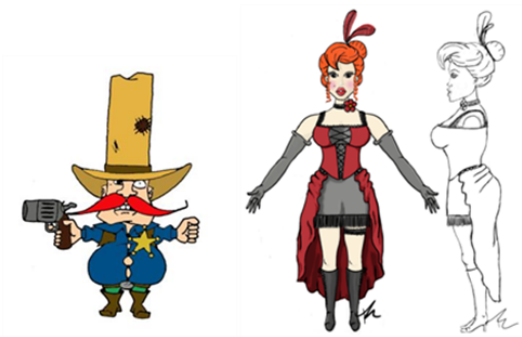


Figure 4: Character concept art of a sheriff and a barmaid model.

Concept artists worked primarily on the main assets of the game (characters and environment), gradually enhancing their work guided by feedback of the art direction and other artists. After the initial sketches on paper, some artists moved on to graphics editing tools, using tablets as input devices. As final step of the concept phase, they created colored front and side views of the assets, which modeling and texturing artists used as guides for their geometry and coloring. Figure 4 shows a small selection of colored character concepts. Most of the concepts were hand drawn and can be found on the development blog.[3]

3.2.2 Low Polygon Modeling

The first step from concept to 3D model is to create a low polygon model. Since Unity can handle triangles and quads, there was no restriction to use a certain modeling technique only. Nevertheless, most modeling artists preferred quads over triangles. Low polygon modeling also includes smoothing or hardening normals.

To avoid models with very differing level of detail, it is highly recommended to define a maximum polygon count in advance, depending on a model's size and importance. However, multiple revisions were necessary during this phase to achieve a consistent style for all the low polygon models.

For houses, a modular system was used to create numerous different house types out of individual basic parts. This tile based approach allowed more variations with less time for modeling.

3.2.3 UV Layout

Before texturing or high polygon modeling could start, UV layouts had to be done either by UV layout artists or by the modeling artists themselves. A UV set in Maya consists of

[3] http://twoday.tuwien.ac.at/mayakurs22010/

a single UV layout. An object with multiple texture types can have multiple UV layouts. For this project a maximum of three UV sets per object was defined:

- Color map UV set: For color textures. Faces can reuse texture space, meaning that a certain space in texture space can be used by multiple faces.
- Normal map UV set (optional): Only used if a separate normal map was required.
- Light map UV set: Can not include overlapping faces in texture space. Since each face can have different lighting information, each face has to have its own amount of space in the UV layout.

3.2.4 High Polygon Modeling

After the UV layouts were done, some of the low polygon models were improved to high-resolution models either by a smooth-operation provided by Maya (adding additional subdivision polygons), or by artistic sculpting using Autodesk Mudbox [5]. The result of this high polygon modeling step was a normal map.

3.2.5 Texturing

Texturing started right after the UV layout was done. A color table (Figure 5) was used as reference for texture artists. A screenshot of the UV layout in texture space was exported to digital image processing tools. The texture coordinates (UVs) served as orientation for texture artists, who painted their textures on these coordinates, using only colors from the given color table.

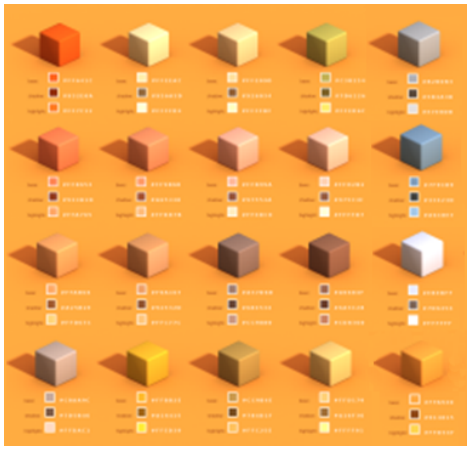


Figure 5: Color table for texture artists

For frequently used textures (such as wood), texture templates were created. For each contained material, these libraries provided bump maps, normal maps and differently colored layers which artist could switch on or off, to create new textures fast and easy.

3.2.6 Rigging and Animation

To enable artists to easily create naturally looking animations of characters, a technique called 'rigging' is used, which creates a virtual bone structure for each character. This structure can then be used to control the movements of the characters for animation in a natural way. After the bone structure is finished, the character's mesh is bound to the skeleton and weighted between influencing bones. Usually, inverse kinematics are used to specify the bones position when moving handles of the skeleton.

In this project, each game character went through an individual rigging process in Maya, offering different controls for the animators. Since this was an educational project, most of the participants decided not to use premanufactured rigs (such as the built-in Full Body IK [3]). The animation artists then produced key-frame based animations for some movements (e.g.: walkcycle, idle, attack) for each game character, which were done in Maya.

3.3 Level Design

3.3.1 The Level Design Process

The previous section described how single pieces of static and dynamic content have been created. The following sections will explain how all these parts are combined to form a virtual environment and finally a playable game. The first step on this path is level design which can be seen as an assembly process that incorporates or creates content to shape the environment of a game. Depending on the genre of a game, the scope of level design may lie on the content creation, the technical assembly, or the game-play tuning aspect.

Scheduling: In general, the crucial amount of work in level design is carried out at a late stage in the creation process. Not only because level design requires parts of the engine to be finished but also assets to work with. In order to allow the level design process start early, it is good practice to prioritize the assets depending on their importance for the level designer.

Concurrent working: Concurrent working on game levels is difficult to be done efficiently and can easily result in a high organization and merging overhead [11]. For small game environments, one person working on the level design is advisable to reduce the overhead. For larger projects, concurrent work of several designers on levels of high complexity can be achieved by splitting the levels along certain regions.

Asset placement: Open terrain game levels often require a lot of environmental objects to be placed into them to produce a rich and exciting game feeling. Automatic placement can help speeding up the process, while keeping a human designer in charge of where objects are placed. For

large projects, more advanced level-editing tools can be used, which allow modifying the layout in a more abstract and faster way (e.g. drawing a map of the area, while the content is placed automatically).

Figure 6 shows the level design approach in relation to other tasks during the creation of a game. The solid lines describe the path of content as it is passed through the team during the game development process. The dashed lines denote requests and tools which will not be included directly in the final release. As shown in Figure 6, level design is a step between content creation and programming. Therefore, changes in level design require interactivity or fast compiling mechanisms to see the impacts on the game.

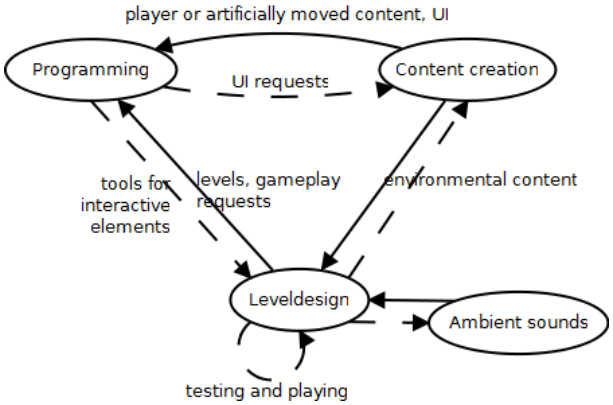


Figure 6: Level design in context to other game development tasks.

3.3.2 Level design in this project

The level design started right from the beginning of the project with concept plans of the game area [8] followed by modeling and texturing of the terrain. After this initial phase a level file was created and assembled in Maya, using references, so that unfinished content (e.g. low polygon models) could already be placed in the scene and be updated on the fly. In the end the whole scene was imported into Unity.

Figure 7 shows the stages of development of the level from the concept art over the terrain model to the final level rendering with buildings and other assets. The level design process took the following considerations into account:

- During the creation of the terrain, conceptual areas were planned for later object placement. (e.g. graveyard area, scenic overview, spots for buildings along the road).
- The level design was based on static geometry. To keep the player interested in exploring the level, many different floors and long paths along the terrain were created.

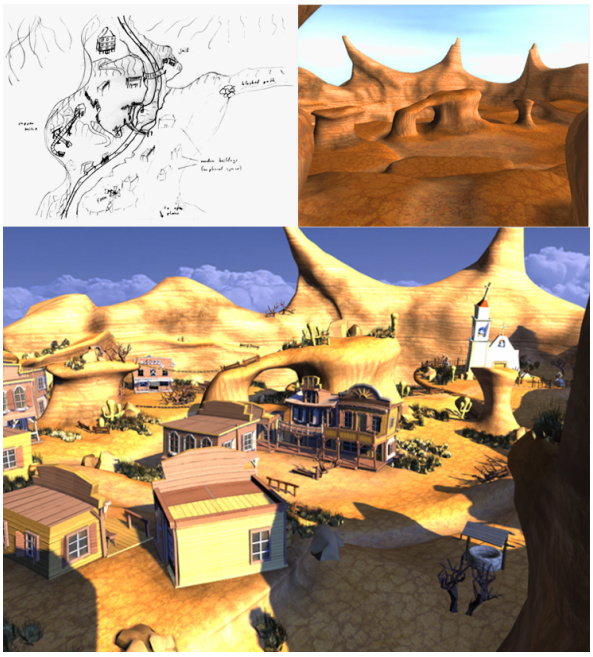


Figure 7: Three stages of level design: concept, terrain model, final scene.

- To underline the escape scenario of the game, a certain degree of hostility was added to the terrain design (e.g. in the shape of spiky rock formations).
- To keep the level simple for lighting, no caves were created.

3.4 Lighting

3D Production packages include powerful renderers to simulate light propagation through a scene. To this point, physically correct simulations of light are impossible because of measuring uncertainty and the inability to solve the rendering equation exactly. In addition, approaches close to physical models are often very costly and cannot be done in real-time yet.

Since computer games are real-time applications, preprocessing for global illumination is often used ('Light Baking') to achieve realistic lighting at runtime. During the process of Light Baking, the whole scene is pre-lit and the final color distribution is stored in textures. These textures can be either applied directly to a polygonal surface or saved as vertex colors. The advantage of Light Baking is that the artists can use high quality rendering algorithms to pre-calculate the lighting of a scene. Disadvantages are the added complexity in the production workflow and the lack of dynamic lights. Combining pre-calculated and real-time lighting effects can be a challenge, because numerous customized shaders and materials need to achieve acceptable color- blending effects.

3.4.1 Lighting in this project

The scene was meant to be illuminated by the sunset with dominating pink and purple colors. Dark areas were avoided to keep the lighting simple. Color textures had to be desaturated, so the scene could be lit with enough power to produce a believable result. Autodesk Mental Ray [6] was used to render the scene. Its fast final gathering algorithm in combination with a light emitting sky-dome was chosen to achieve the effect of global illumination for the game's content.

The sun consisted of two simple directional light sources not affecting the sky-dome. One light-source was used for controlling the light color and intensity. The other light source was used for controlling the shapes, intensities and colors of the shadows.

It was important to know the scene's look under final lighting conditions at an early stage of the production process to define appropriate texture color guidelines. Therefore, this light setup was developed at an early stage to test different texture colors and shaders in the chosen environment setting.

The final step in lighting was light map texture baking. Diffuse color bouncing required emitting and reflecting colored objects in the scene during bake-time. These objects had to be prepared for baking. First, they were given a second UV-set, which was not overlapping or tiled.

For light-mapping and export, objects were grouped into export groups, sharing one common light-map, and were then imported separately into Unity. A building and parts of its corresponding light-map are shown in Figure 8.

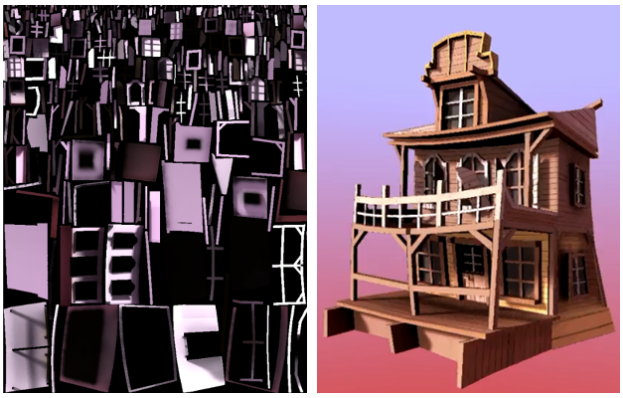


Figure 8: Building to the right and its corresponding light-map to the left.

3.4.2 Concept and Presentation Rendering

In addition to the scene's lighting, rendering was also used for many different tasks during the development process. Concept renders were created in order to communicate the aesthetics of the game environment. A game trailer was produced showing some animation sequences and renders were created by artists to share their concepts and models through the team's development blog. Since rendering is a time consuming process, some additional tricks were used to improve quality in these renderings. Depth of field effects were achieved through depth-map controlled blur filters using compositing. Additionally, image post processing was used to achieve the desired quality.

3.5 Implementation

During the implementation phase, intensive work with the game engine started, as all the models had to be imported into Unity. Unity generally uses *.fbx files, but allows the import of *.ma files as well. These Maya files were directly imported into Unity to avoid problems with animated objects exported by Maya 2011 as *.fbx and to skip an additional step in the workflow. The files were then converted into *.fbx by Unity. This procedure makes it easy to re-import changed Maya files but it takes more time for Unity to convert all files.

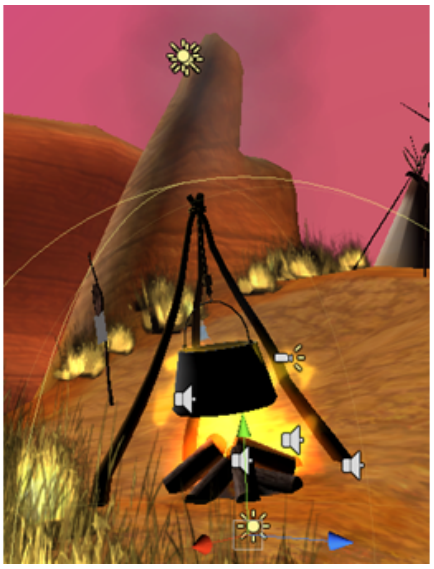


Figure 9: Fire, smoke effects and sound sources.

3.5.1 Object Integration in Unity

In the first step of the object integration process the level, which had been created in Maya 2011, was imported into Unity. It was structured into different display layers, each one dedicated to a different type of models (e.g. houses, grasses, cacti). When the level was completed, the different layers were saved in separate Maya files and put together in Unity. This made it easier and faster to re-import changed layers into Unity. The different characters as well as various pickup items (e.g. dynamite) used in the game story, were directly placed into the level in Unity.

3.5.2 Effects, Shader Programming

As soon as the main part of the game was finished, effects like fire and smoke (Figure 9), fog, water, dust, lens flare and explosions were implemented. All these effects, except lens flare and fog, were created with an Ellipsoid Particle Emitter or a Mesh Particle Emitter provided by Unity. Fog was simply activated in Unity's render settings.

3.5.3 Sound

In the final phase background music and sound effects were added to the game. To add a sound file to an object in Unity, an 'Audio Source Component' had to be added to the specified object. In Unity's audio source options, the audio clip had to be defined and some other options like the volume were individually adapted.

3.6 Release

The final result of the whole production process was a game prototype with one level. Overall, it took a production team of 26 students about 1550 person hours to develop the game 'Dynamite Pete'. Table 1 gives a detailed overview of the working time spent on the individual phases of the project.

Task	Time in person hours
Concept and references	130
Low polygon modeling	500
UV layout	150
High polygon modeling	50
Texturing	190
Rigging	20
Animation	50
Level design	50
Lighting	60
Implementation	200
Miscellaneous (presentation)	50
Organisation and feedback	100
Total	1550

Table 1: Development time overview

After the end of the project, the final game was presented in a release event. The artists created a trailer video, character sheets, posters and character turntables for this event.

4 Further Challenges

4.1 Terrain Multi-Texturing

The usual approach for texturing 3D models is to apply a single texture per object. Since the terrain is a very large object, tile-able textures were used to define its surface properties. These textures are repeated seamlessly across a geometric surface representation. However, since terrain usually consists of different material, more than one tile-able texture had to be used.

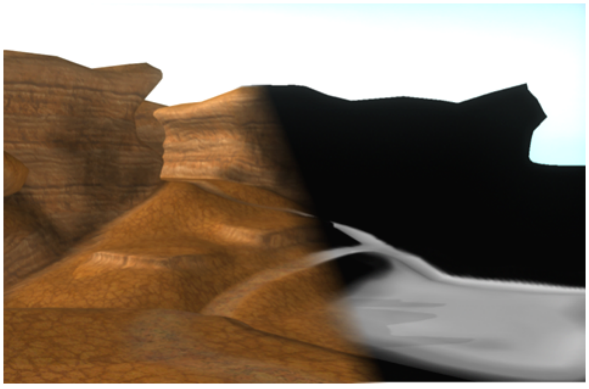


Figure 10: Left side: textured terrain. Right side: alpha map for the path layer.

Multiple layers of tile-able textures were used in combination with an alpha texture to define the global occurrence of a material. Figure 10 shows the textured terrain on the left side and the alpha map for one layer on the right side. In Maya, the material that supports this kind of texturing method is called 'Layered Texture'. In Unity, this method of texturing is only provided for terrains created from height-maps. Therefore, a custom fragment shader had to be written for the terrain in this project.

The following shows the essential part of this fragment shader code. First, the albedos of the individual overlay layers are consecutively accumulated. Finally, the calculated albedo is multiplied with the light intensity to obtain the output color.

```
//'c', 'c1', 'c2', 'c3' ... color textures
//'a' ... alpha values in rgb channels
//'lightmap' ... light map texture

//overlay 1st layer
o.Albedo = (1.0f-a.r)*c.rgb + a.r*c1.rgb;
//overlay 2nd
o.Albedo = (1.0f-a.g)*o.Albedo + a.g*c2.rgb;
//overlay 3rd
o.Albedo = (1.0f-a.b)*o.Albedo + a.b*c3.rgb;

//1.5f to control light map brightness manually
o.Albedo = o.Albedo * lightmap * 1.5f;
```

4.2 Concurrent working with Unity

As mentioned earlier, there are some differences between Unity Free and Unity Pro. Because the game was developed with Unity Free, a solution for the SVN problem had to be found. Unity Pro saves all asset metadata and import settings for each asset in a corresponding metafile. These files need to be versioned along with the associated asset

[2]. In Unity Free all these metadata are saved in the library directory [1]. To share a project without Unity Pro, the library directory has to be compressed and uploaded via SVN. After downloading the project from the SVN server the library directory has to be uncompressed into the Unity project folder. One negative effect of this solution is that it is not possible to work concurrently on the Unity project file. Because the whole library folder is uploaded, it is not possible to merge the different metadata, only the whole folder can be overwritten. Therefore, for multiuser projects it is recommended to use Unity Pro.

5 Summary

Together with our colleagues, we deployed a professional game development studio workflow in the context of a 3D content creation project. The utilization of 3D content in a 3D game put theoretical knowledge into practical use. Participating students improved not only their technical skills, but also learned valuable lessons in team work. The major challenges were different theoretical and practical understanding of the subject, unalike working methods and variable personal availability. A blog was used by most of the students to post their work and to get feedback in a constructive way. The final stages in the process of game development were rushed in about one or two weeks. At the time of the release deadline, the result was a decreased quality in game-play and lighting as well as the game mechanics, which had not been tested before. More concurrent work and earlier deadlines would have been needed to speed up the production process. Nevertheless the project was finished within the given time schedule, resulting in a nice game containing rich and detailed content.

6 Acknowledgements

The authors would like to thank all team members who participated in this valuable exercise and dedicated their time to this project. Thanks to Rainer Angerer, Martin Brunnhuber, Damir Dizdarevic, Brigit Faber, Markus Fellner, Roman Gurbat, Andreas Himmetzberger, Rosemarie Hochreiter, Roman Hochstöger, Bernhard Holzer, Peter Houska, Albert Kavelar, Desiree Lavaulx-Vrecourt, Andreas Lenzhofer, Thomas Mayr, Johannes Sorger, Stefan Stangl, Nicolas Swoboda, Markus Tragust and Ulrike Zauner. Our special thanks to our lecturer Markus Weilguny, who guided us through the whole production process. Last but not least we would also like to thank Associate Professor Michael Wimmer for making this lecture possible and for giving us the great opportunity to present our efforts and findings in this paper.

References

[1] Unity 3D. Unity: Behind the scenes. http://unity3d.com/support/documentation/Manual/Behind%20the%20Scenes.html, 2010.

[2] Unity 3D. Unity: Using external version control systems with unity. http://unity3d.com/support/documentation/Manual/ExternalVersionControlSystemSupport.html, 2010.

[3] Autodesk. Autodesk maya online help: Fbik (full-body ik). http://download.autodesk.com/us/maya/2010help/index.html?url=Glossary_F_FBIK_fullbody_IK.htm, topicNumber=d0e188540, 2000-2009.

[4] Autodesk. Autodesk maya 2011: Features. http://usa.autodesk.com/maya/features, 2011.

[5] Autodesk. Autodesk mudbox 2011: Features. http://area.autodesk.com/mudbox2011/features, 2011.

[6] Autodesk. Mental ray. http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13566140, 2011.

[7] Disney/Pixar. Toy story. http://www.pixar.com/featurefilms/ts/, 1995-2011.

[8] Michael Stuart Licht. Gamasutra: An architect's perspective on level design pre-production. http://www.gamasutra.com/view/feature/2848/an_architects_perspective_on_, 2003.

[9] Lucasfilm Ltd. Star wars. the clone wars. http://www.starwars.com/theclonewars/, 2011.

[10] Unity Technologies. Unity 3d. http://unity3d.com/, 2011.

[11] Mick West. Gamasutra: Collaborate game editing. http://www.gamasutra.com/view/feature/3991/collaborative_game_editing, 2009.

Human Computer Interfaces

Multiplatform framework for managing windows

Michal Kevický*
Supervised by: Silvester Czanner†

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

Abstract

We introduce a multiplatform C++ framework for managing application windows respectively displayed content. The framework provides an API for handling display devices, representation and presentation of displayed content, managing input events and other actions. Furthermore, objects are extended with semantic context which defines their functional properties. Semantics allows more accurate classification of entities, choosing more accurate reactions for incomming signals. The framework provides platform, which does not strictly rely on classical WIMP (windows, icons, menu, pointer) paradigm and gives ability to develop solutions non-conventional way (e.g. zoomable UI, tangible UI).

Keywords: Graphical user interfaces, Windowing systems, Frameworks, Scalability

1 Introduction

Lately we experience vast expansion of electronic gadgets with many different graphical user interfaces. Each new interface introduces new philosophy of control, thus confusing users and leading to deformation of user's learning curves of such systems in a negative way. It increases the costs of the development of multiplatform applications and brings new possibilities for various errors.

Graphical user interface usually depends on 4 parts. The first part, an *operating system* (**OS**), is not the one which should be important for real layout. OS is here to provide communication with *input/output* (**I/O**) devices, management of resources, communication between processes, and handling permissions, and entire user model. OS defines a "platform". The second part is a window managing application (i.e. *window manager*, **WM**). It manages the content and defines how should this be displayed on attached displays. Windows management is the main contribution of our work. The third part is represented by the content of windows. It usually consists of a canvas containing set of well positioned widgets. Since there are many multiplatform widget sets (e.g. wxWidgets, Qt, GTK, Swing, AWT, ...), there should be no problem of designing and implementing portable layout of window. The fourth part is represented by input devices, i.e. physical interaction, is the part which can not be modified software way and hence also is not field of our interest.

WMs are either part of OS (like in Windows series from Microsoft) or can be separate application like in many of *NIXes. As a bonus, in *NIX X Window system, each application can act like a WM, what caused there are really a plenty of WMs now.

WMs of present days usually do not provide interface for customization/extension in non-layout properties (i.e. in behaviour, input devices, representation of windows) using predefined APIs, leading to workaround solutions. Our solution implements modular framework model which is easy to extend and modify. An end user for our framework is primarily WM or presentation logic programmer.

The framework is here to define and implement abstract interfaces for the end user to ease writing of portable respectively multiplatform code. The implementation of the framework for a specific platform consists of *core* code of the framework and set of the *platform specific modules* or *wrappers* providing functionality defined by the abstract interfaces mentioned above.

The structure of this paper is organized as follows:

Section 2 introduces few existing projects implementing similar functionality or providing relevant informations to our work.

In Section 3, we describe used technologies, reasons for their selection and their pros and cons.

Section 4 describes architecture of framework, object structure and application flow.

Section 5 presents advantages of framework and possibilities it brings for window manager / application programmer and end user of final window manager.

Section 6 sumarizes our results, section 7 presents our plans for future work and section 8 provides conclusion of

*kevicky@gmail.com
†s.czanner@warwick.ac.uk

entire article.

2 Related work

In the world of WMs, there is a large scale of a different applications or application packages, from lightweight programs to complex desktop environment packages. Especially in world of *NIXes, there are hundreds of options to choose WM. Here is an overview of few works interesting to us:

The *Compiz-fusion* [3] project is alpha for recent Linux world in field of desktops and visual effects. The Compiz-fusion project implements using AIGLX/XGL extensions to the X Windows system an compositing windows manager introducing different visual effects (wobbly windows, transparency, desktops mapped on rotatable cube, ...) to the world of Linux.

The *Metisse* [8] is French project extending FVWM project which has brought their own implemention of composite extension and has come up with their modification of X server. The goals of the project are similar to ours - easing the design, implementing and testing of new *human computer interaction* (**HCI**) approaches and techniques. What is worse, the Metisse is primary X Window System oriented. On the other hand, one of goals of our framework is the development of unified multiplatform base for managing windows.

The *SphereXP* [5] is Slovak project by Dušan Hamar, and it has been one of the pioneers of Microsoft Windows XP window managing customization. SphereXP provides extension of standard flat rectangle desktop to space inside sphere. Together with KDE project and Blackbox, SphereXP is the one that proves, that customization of the presentation layer of Windows XP is really possible.

Last but not least there are *Blackbox* [1] and *Fluxbox* [4], the fork of Blackbox. Both are very lightweight WMs implementing stackable WM concept, providing spartan visual interface. For minimum functions they provide, and MIT license, they are released under, they has became inspiration and basic code of our framework.

The Quartz-compositor as part of MacOS X by Apple or the Aero from Windows 7 by Microsoft are well known technologies so they will not be discussed.

3 Used technology

Response times of the visual interfaces are one of their critical features, so it is expected to use speed efficient language. WM can also be considered as feature from "system programming" category. On the other hand, framework is here for other users to simplify them implementation. The C sometimes appears to be one of the synonyms for system programming, but also it is not so simple as Java or C#. Compromise between the power and the userfriendliness seems to be combination of the *C* and the *C++* with possible *inter process communication* (**IPC**) with modules written in different languages.

However *Portable Operating System Interface [for Unix]* (**POSIX** IEEE 1003 standard [6]) is primarily oriented to the *NIXes, and it should be considered as a guideline for the multiplatform computing.

We have used *Boost*[2], because it is set of peer-reviewed C++ libraries which are almost the part of the upcoming C++ standard. It also comes here to provide rapid application development features.

Ontologies and *description logics* are concepts known from semantic web. Main reason for using them is a possibility of describing objects by their properties, where classification of objects using taxonomies faces problems. These entities should act like good candidates for multiple branches of a tree. Except for the problem with multiple occurances in the taxonomy tree[1], description logics solve also the problem with resolving context and scalability. For example, there is no need for launching a particular multimedia player[2], there is only need for a player capable of playing 7.1 sound, respecting DRM[3] as well as the possibility of controlling with a mobile phone using a bluetooth technology.

4 Architecture

WMs can be divided into 3 categories: the tailing WM, the stacking WM, the compositing WM.

The *tailing* WM splits planar screen into the disjoint zones (with no visual overlaping between applications). This concept has been used in earlier versions of Microsoft Windows (e.g. version 1.0).

The *Stacking* WM introduces the layers of windows, allowing overlaping of windows. This concept can be found in classical WMs running on X Windows System, including Blackbox and Fluxbox, KDE to version 4, also Microsoft Windows prior to Microsoft Aero technology, and Mac OS prior to OS X. Main difference between the *stacking* and the *compositing* WMs is, that stacking ones draw content of windows directly meanwhile compositing ones let application to render their output to buffers and the WM work with content from WM's buffers. Representatives of the compositing WMs are the Apple's Quartz-compositor, the Compiz-fusion project or the Microsoft's Aero. The primary goal of the framework was to develop the compositing window manager, but it

[1] one object could belong to multiple tree branches, which should be disjoint

[2] such a player might not even exists on target system

[3] digital rights management

is not problem to customize it for the old school methods mentioned above.

Most of the recent WMs which are implementing standard WIMP paradigm user interface suffers with several limitations. The first to mention is, that the object's structure has fixed amount of properties (e.g. window: dimensions, position, title, scalability, min/max/normal state and that is almost complete listing). There is no native way to communicate any different properties between the applications.

As WMs assume 2D windows, recent controls has also been designed for 2D interfaces. 3 (and more) dimensional interfaces are just part of sci-fi or experiments, but there is not native way for integration into the WM's system. Similar situation occures while considering displays - 2D is standard, 3D is sound of future.

The framework solves this problem with *dynamic amount of properties* of the "window" object. Window object itself is quite confusing notation in the context of the framework. The framework itself contains "content" entity which the most important property is the binary blob. The binary blob contains only "some kind of data". Other relevant property is dataType which determines, which interpreter to use (does binary blob represent pixmap, plain text, h264 video stream, set of 3D instructions?).

Our framework consists of *core*, which defines basic structure split into the four layers (*figure 1*): *Input layer* (**IL**), *Application logic layer* (**ALL**), *Presentation layer* (**PL**) and *Client application layer* (**CAL**). Other important parts of core are abstract interfaces and functions for management of modules and communication between them and the platform specific wrappers.

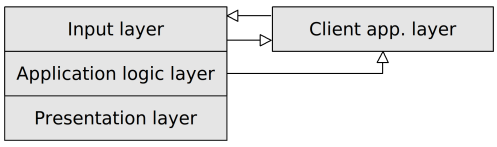


Figure 1: Framework's core

The *input layer* (*figure 2*) handles input events received from the OS, the client applications, or events generated inside the system. Events are being classified, passed to the priority queue, subsequently routed with respect to the security policies to different queues - high priority for realtime applications and slower one with ontology mapping for future pass into *application logic layer*.

The *application logic layer* (*figure 4*) is the brain of entire framework. Here is the place for modification of appli-

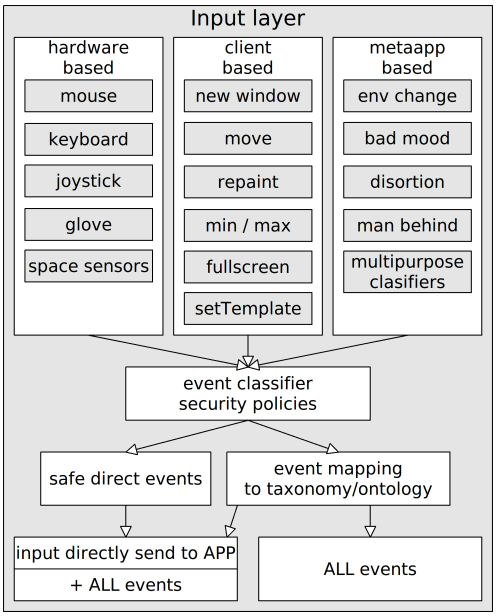


Figure 2: Input layer

cation logic, write own rules, generate events and change configuration of state machine. ALL configures actions for PL, passes commited events to client applications (i.e. "windows"). Aplication logic is designed to be fully modifiable and portable, as far as it operates with the abstract entities. At the time, only conservative processing engine is fully implemented. It processes events by executing code in written form (i.e. C++). Another available concept for processing unit is using visual modelling - in our case there is the modified *timed petri network* (m**TPN**).

TPN consists of *places* and *transitions* connected with *directed arcs* (each arc connects one place with one transition). Places acts as accumulators of tokens - requests. Transitions represents actions or parts of code. They can be enabled or disabled. Directed arcs represents set of conditions that have to be satisfied (accumulated sufficient amount of tokens) to enable transition. If conditions are not satisfied, transition is disabled. Enabled transition have to fire (execute code) in defined time conditions (i.e. it have to fire from t_{from} to t_{to} time units since transition has been enabled). There are various modifications of Petri Networks, introducing different enhancements like mentioned time, different types of tokens, spetial additional conditions. Petri Networks are also designed for verification of algorithms, detecting deadlocks, infinite loops etc., which are not considered in this stage.

Modeling with TPN is illustrative enough for people not experienced in programming but with mathematical aparate. Visual modeling utility for mTPN is part of future work.

ALL can be also extended with any other visual mod-

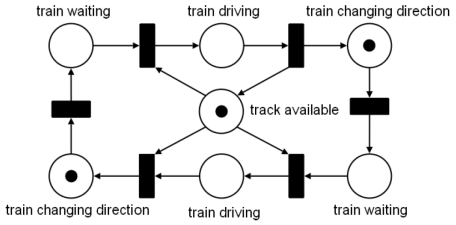


Figure 3: Petri network sample: places (rings), transitions (rectangles), tokens (dots) [7]

eling system[4], but we are not planning to implement any others.

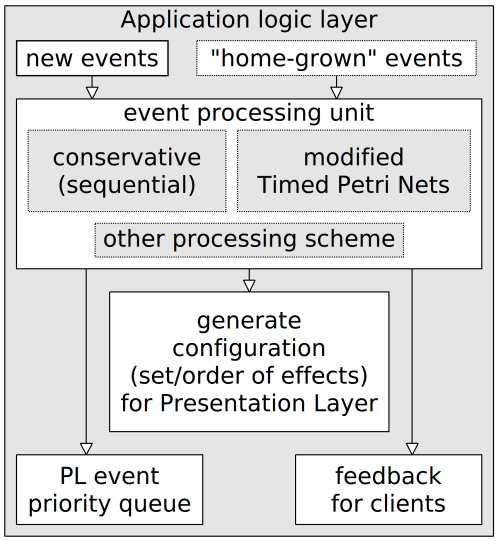


Figure 4: Application logic layer

The *client application layer* provides interface between the applications outside the framework (clients) and framework functionality. This should be done in two ways. The first is done by the wrapper, which does not need cooperation from client, wrapper just parasites on its interfaces and translate framework's communication to platform native codes and vice versa. The second is to handle communication native way, i.e. write "window" application using framework's API, what makes the client application framework dependant. Using framework's API may be useful for example if a user needs quickly gain existing feature. As example should be simple image viewer, which only initializes framework's communication object, sets dataType to "image/jpeg", and loads image into the attached binary blob property. The rest is work of PL and framework's interpreters/renderers.

The *presentation layer* is processing configuration prepared by the ALL. PL is the one responsible for visual interpretation binary blobs (by renderers), preparing scene,

[4]like timed automata etc.

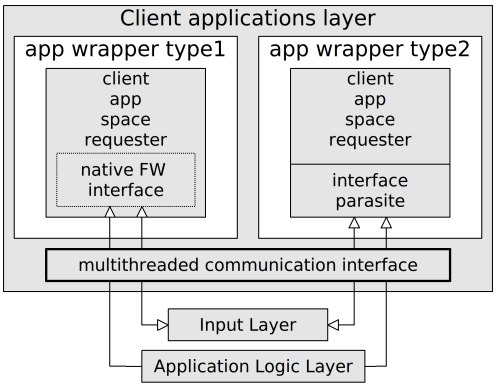


Figure 5: Client application layer

caching scenes, work with hardware acceleration, buffers, decorating output with special effects, finalization of rendered scene. PL is also place for user's own modules for special effects and visual data interpretation.

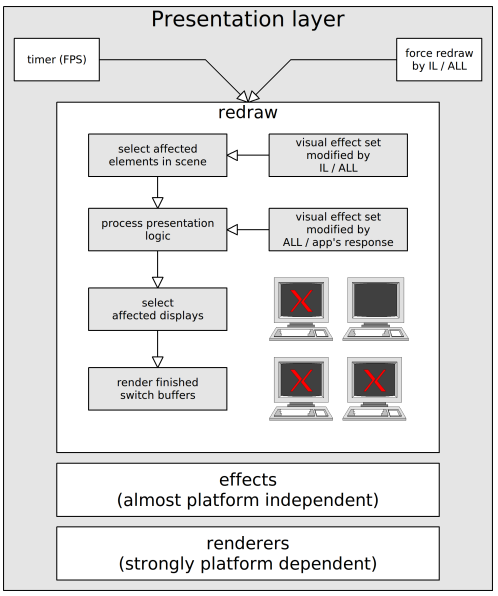


Figure 6: Presentation Layer

5 Possible applications

In fact framework separates output from application, what gives us flexibility for further manipulations. Several examples are mentioned here, but they are just examples, they should look simple but are there mentioned just for illustration. Not every of modules is implemented at present time, few are part of future work.

5.1 Image processing

As mentioned above, a visual output from application is just a binary blob, which could but does not have to be

passed for further graphical/data processing. For example, there is need to display histogram of a 24bpp[5] bitmap. Once a module implementing that functionality is implemented, programmer just need to load it and add to processchain for specific window and application developer does not need to implement it again (i.e. reuse of code).

5.2 Distributing output

Can be useful for viewing and recording video at same time - one machine can act as a high quality display, another as a recorder/encoder and last one as a streaming server for the internet (i.e. load balancing).

Another example is sharing applications by extending paradigm of a remote desktop. Machine A is running application and renders output, machines B and C are displaying same output window for two different users and are receiving inputs from them, both B and C are passing input signals back to A (i.e. collaboration).

5.3 Space

Most of recent WMs are still 2D and single machine oriented. Framework does not rely on any number of dimensions or machines. If there is need of 3D interpretation of a content, there is possibility to add property[6] of depth and Z coordinate of position to each window and write logic, which operates in 3D space. If there is need of union of vritual space of multiple machines, there is possibility join them into virtual domain.

5.4 Application classification

Each person has it's own organizational structure for objects by their characteristics. Dishes, food and things associated with eating are usually stored in kitchen. Also applications can be grouped to categories - work, social interaction, relax, etc. . This can also lead to organizing them in space, giving several groups of applications different priorities in interaction and machine time. Switching between applications (usually by pressing *Alt+Tab*) can also be extended with groups, like keys on different keyrings and user can quickly switch between "windows" in specific group and does not have to traverse unimportant ones.

Frequently popuping applications can also receive less priority by classifiers in ALL to surpress disturbing effect. Multiple behaviours of entire interaction systems can be introduced based on user's settings or for example user's mood (switched on requests or by receptors).

5.5 Kiosk mode

Or restricted (interaction) environment is often used for single purpose applications installed in a public or open space, where a machine should serve only one specific application and should be resistant again attempts to gain unprivileged control of a machine. Building restricted environment is not a simple task, which usually requires sanitizing of input systems. Customization via framework is simple task, because can be switched to mode when no other application can be displayed or receive input from user.

[5] color depth 24 bits per pixel

[6] dynamic properties vs. limitation of existing WMs

6 Results

We have developed the core and basic parts (i.e. systems wrappers) of framework. Actual port is working on 32 / 64bit Linux versions, port to Windows XP is in progress at the moment. In actual state of development, porting the framework to a specific platform is mainly work of porting wrappers to OS + platorm specific utilities to handle communication with drivers (including renderers, wrapper for CAL as shown on figure 7). At the moment, framework is not oriented to high performance.

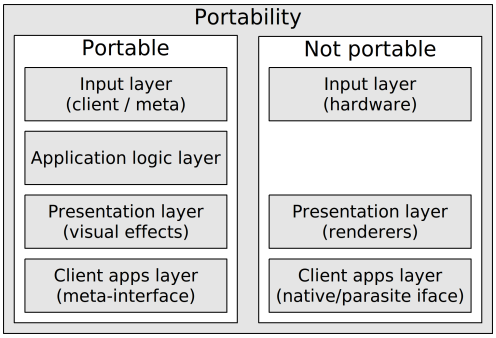


Figure 7: Portability

7 Future work

Future development can be divided to

- a visual modeling tool for mTPN as well as pre-fail verification for modelled mTPN structures
- visual configuration utilities
- porting to mobile devices
- network functionality, visual domain - a space of a rendered output shared between multiple workstations in domain
- improving performance
- improving security

8 Conclusion

Basic structures of concept were successfuly implemented, but there is still place for improvements. Our interest for future work focuses to mobile platforms, improving configuration interface and implementing mTPN visual interface to ease interaction for common people.

References

[1] Blackbox, February 2011.
http://blackboxwm.sourceforge.net/.

[2] Boost c++ libraries, feb 2011.
http://www.boost.org.

[3] The compiz-fusion project, February 2011.
http://www.compiz.org/.

[4] Fluxbox, February 1. 2011.
http://fluxbox.org/.

[5] Dušan Hamar. Spherexp, feb 2011.
http://www.spheresite.com/.

[6] IEEE. *IEEE Standard Interpretations for IEEE Std 1003.1-1990 and IEEE Std 2003.1-1992*. IEEE, 1994.

[7] Jan Krumsiek. Petri nets for network analysis, 2007.
http://www2.bio.ifi.lmu.de/lehre/SS2007/SEM_Advanced/Folien/petrinets_slides.pdf.

[8] Nicolas Roussel Olivier Chapuis. *Metisse is not a 3D desktop*. UIST '05 Proceedings of the 18th annual ACM symposium on User interface software and technology, ACM New York, NY, USA , 2005.
http://insitu.lri.fr/metisse/.

Multi-touch Table with Image Capturing

Jakub Hušek[1]

Supervised by: Miroslav Macík[2]

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

Abstract

This text reviews the state-of-the-art of technologies for multi-touch control and taking photographs or video recordings through the screen. Furthermore, the work describes development of a prototype device combining a multi-touch screen in a table setup with the ability to take pictures through the viewing area.

Keywords: Human-Computer Interaction, Multi-touch, Switchable Diffuser, Scanning

1 Introduction

In recent years, the development of computer technology experienced rapid expansion into all various areas of industry and everyday life. Hence the importance of natural ways of interaction between humans and computers in all forms is growing strongly. There is a demand for interfaces that are intuitive (and thus the learning curve is steep for their use) and which allows users to employ theirs potential and do not needlessly slow down their activity. Assessment, which user interface is appropriate in different use-cases is usually matter of research.

The goal of this work is to implement a prototype of an interactive device for such research. Specifically, it is an interactive table that combines a touch screen with the ability to take pictures (scans) of objects through the imaging area.

Following scenarios illustrate possible motivation for developing this method of interaction. For instance, imagine a designer, who draws a draft for a new logo on a piece of paper. He comes home to his interactive device, puts the paper on the screen and the drawing is scanned. He can instantly interact with his draft in a digitalized form using the computer. Of course, he could use scanner, but the case mentioned is more straightforward and intuitive approach.

Another example: interactive kiosks are located around a large hospital. A patient comes to see a doctor. She has no idea about the actual position of the doctor's office. But she has an invitation card with a barcode. She comes to the interactive kiosk, which instructs her to place the card on the surface of the kiosk. After that, the barcode is recognized and she is navigated to the right place.

Finally, tangible object, such as cell phones, cards, chessmen, etc. lying on an interactive surface, can be recognized easily and they can be used as controls.

However these examples are only suggestions. Proper research must be done to determine, if these and other similar use-cases are really usable and worthwhile. The purpose of our prototype is to allow such research.

2 Related Work

This section discusses several technologies, which can be used for our device. The first part focuses on a multi-touch control. The second part summarizes technologies for capturing image from the screen direction.

2.1 Multi-touch Input

Touch input surely is a promising way to bring more naturalness and real behavior into the human–computer interaction. That is why there has been a great attention recently [1]. Interest in touch technologies among the public increased in the 2006, when Jefferson Han introduced his multi-touch display based on FTIR technology [2] (see 2.1.4). After that, many people built their own tables with multi-touch screens. In the beginning of the 2007, Apple introduced theirs multi-touch cell phone named *iPhone*. It used capacitive technology. The *iPhone* affects whole segment of the mobile phone industry. In the same year, *Microsoft* introduced the *Microsoft Surface* – a coffee table with a multi-touch screen. This is based on the optical method *Diffuse Illumination*, which is described later in 2.1.4.

In general, touch input provides not only information about positions of contacts with the surface, but also about the relative pressure or angle of the touching object (especially a finger) [1]. Technologies vary in abilities of different objects detection. Some can detect only fingers, and do not allow usage of a stylus or other objects. Some technologies allow tracking of close objects, which are not really touching the display [1]. Finally, there can be a demand to identify the objects lying on the display. It can be achieved by reading special patterns (tags) glued to the bottom of these objects. A summary of particular technologies that can be used for touch input follows.

2.1.1 Resistance Based Technologies

Resistive touch surfaces consist of two conductive layers with a tiny gap between them. When pressing the upper

[1] husekjak@fel.cvut.cz

[2] macikmir@fel.cvut.cz

flexible layer, it contacts with the bottom one. There are two basic methods to determine the contact position [3].

The **four-wire technology** drives a voltage gradient into the first layer and a voltage is measured on the other. This is used for calculating the first coordinate. The second coordinate is acquired analogically by changing the function of the respective layers.

The second method (the **five-wire technology**) attaches the voltage always to the upper layer and measures the current from the corners of the bottom layer.

There follows a list of general advantages and disadvantages of resistance based methods:

Advantages

- any type of object (materials) can be detected
- resistant to dirt, dust, water or light pollution
- low power consumption

Disadvantages

- only about 75% to 85% transparency
- degradation of the upper layer over the time (five-wire technology is more durable)

2.1.2 Capacitance Based Technologies

Capacitance based technologies are available in two basic variants too [4].

Surface capacitive touch panels consist of thin conductive coating on a transparent glass. From each side of the panel electrodes maintain a uniform electric field across the conductive layer. As human fingers (or other conductive objects) are capable of exhibiting electric fields, touching the panel results in a small transport of charge from the electric field of the panel to the field of the touching object. The value of electric current drown from the corners is measured with corresponding sensors. After that, microprocessor calculates an exact position of the touch based on the values measured.

Projected capacitive touch surfaces contains very thin grid of wires or electrodes between two glass layers. The grid behaves as a matrix of capacitors. When touched, capacitance forms between the finger and the sensor grid. The touch location can be computed using the measured electrical characteristics of the grid layer. Depending on the implementation, this technology can detect more touches at once than surface capacitive technology.

Advantages

- higher transparency than methods based on resistance (especially the projected capacitive method)
- high resolution and accuracy
- high mechanical robustness (especially the projected capacitive method – it can be covered with up two centimeter protective layer)
- resistance to dirt, dust, grease
- higher reliability than methods based on resistance

Disadvantages

- higher price (especially the projected capacitive method)
- detects only conductive materials (but it can be an advantage – unintended touches by clothes are ignored)

2.1.3 Acoustic Technology (Surface Acoustic Wave)

In this technology, ultra-sonic waves are induced and directed from piezoelectric transducers in orthogonal directions over the surface [3, 4]. When touched, the measured signals are changed, which is used to calculate a position of the contact.

Advantages

- no transparency limitation
- robustness of construction
- high resolution
- resistance to intense light

Disadvantages

- limited to fingers and soft objects
- affected by dirt and dust negatively
- limited count of touches detected simultaneously

2.1.4 Optical (Infrared Light) Based

Most optical methods are quite easy to implement and much cheaper than capacitance based methods. This determines theirs usage in many, often home-made or academic, prototypes [5]. Additionally, some optical technologies allow more detection modalities as mentioned in the introduction of this section (see 2.1).

Optical methods are usually based on capturing the surface in infrared light (IR). There are several techniques, which differ in the arrangement and in the corresponding features. In general, optical based methods have the following advantages and disadvantages:

Advantages

- very good transparency
- multiple touches detection (depends on a method)
- detection of tags and close objects (some methods)
- resistance to water, degradation
- relatively low price and simple manufacturing process
- recognition of arbitrary objects, shapes (most methods)

Disadvantages

- requires complex computation for touch detection
- sensitive to intense ambient light

In the following, individual optical methods are described briefly.

Light Grid (LED Light Plane)

This technique is very similar to the mentioned acoustic method. Instead of acoustic waves, there is a grid of

infrared light over the surface, emitted by LEDs located around [3]. When touched, the infrared ray is interrupted, which is detected by sensors on the edge and evaluated as a touch.

Advantages

- simplicity
- resistance to dirt and dust
- can be mounted on a conventional display

Disadvantages

- cannot recognize arbitrary shapes, tags and close (but not touching) objects
- cannot recognize pressure
- lower resolution
- bad scalability (many sensors)
- limited count of touches (may occur occlusion)

Laser Light Plane (LLP)

The infrared light from a laser is driven through dispersive lens just above the surface, so it generates a light plane [6]. When touched, the light reflected from the tip of the finger is captured by a camera located below the active surface (see Figure 1) and the position of touch can be determined.

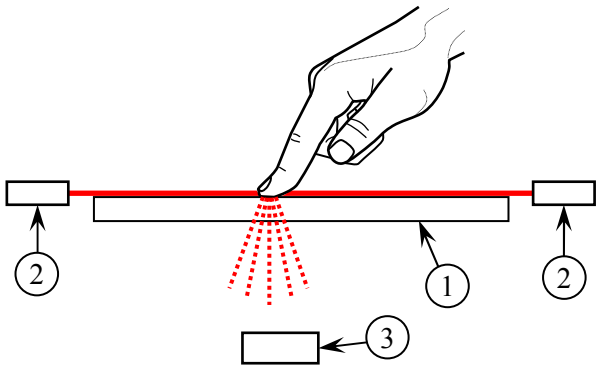


Figure 1: Laser Light Plane:
(1) glass or plexiglass, (2) IR laser, (3) IR camera

Advantages

- simplicity
- relatively cheap

Disadvantages

- cannot recognize arbitrary shapes, tags and close (but not touching) objects
- cannot recognize pressure
- sometimes limited count of touches (may occur occlusion)

Frustrated Total Internal Reflection (FTIR)

This technology – used by Jefferson Han [2] – is based on a principle of total internal reflection: Touch panel made from acrylic glass (plexiglass) is illuminated in infrared from edges of the surface. If the refractive index of the surface panel material is higher than the refractive index of the surroundings and the light incidence angle at the boundary is higher than the critical angle, total reflection occurs, and all light spreads through the material. When an object with a higher refractive index touches the surface, some light leaks, which is captured by an infrared camera behind the panel. The acquired video stream is used to determine position of touches.

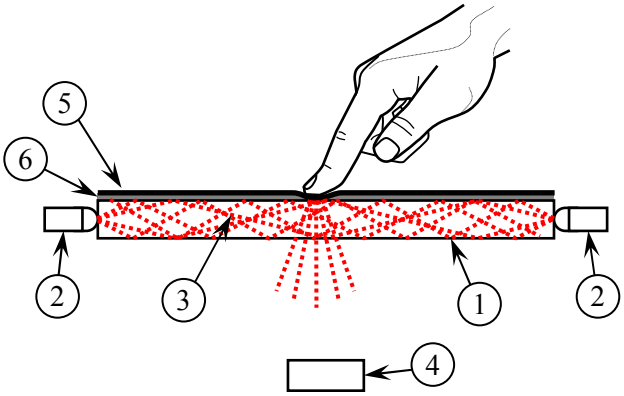


Figure 2: Frustrated Total Internal Reflection: (1) acrylic glass, (2) IR LED, (3) total internal reflection, (4) IR camera, (5) projection layer, (6) compliant layer

There must be a good coupling between the touching object (finger) and the acrylic glass to trigger a FTIR effect. It can be achieved by pressing hard to the surface, but this decreases ease of use. Dragging makes the FTIR effect even weaker [4]. Therefore, there is usually another layer applied – a compliant layer. It is a thin layer of soft translucent material, which is put (or coated) between acrylic and a top layer. The upper surface (see Figure 2) is a diffusing layer, used as a projection plane. While touching, the upper layer couples with the acrylic and triggers stronger FTIR effect. Moreover, it allows interaction with objects of arbitrary material – there are no constraints for the refractive index of the touching object, when using compliant layer. Finding an appropriate material of a compliant layer and the way of its application (in the right combination with a projection layer) is a crucial step of the implementation [7, 8, 4].

Advantages

- recognition of multiple touches and any shapes
- good scalability
- high accuracy and robustness
- can detect pressure indirectly (by size of the touched area)
- with a good compliant surface, it can be used with something as small as a pen tip
- an enclosed box is not required
- usable on non-planar surfaces
- cheap

Disadvantages

- size – depends on viewing angle of the camera(s)
- requires a LED frame
- requires acrylic (regular glass cannot be used)
- cannot detect close objects (not touching)
- complex implementation of the compliant layer

Diffuse Illumination (DI)

This method is based on uniform diffuse illumination of the surface from the bottom [4] (see Figure 3). An infrared camera captures light reflected from objects on the surface. The diffusing layer (which can be used as a projection surface) is important for object recognition. Blurriness of reflected objects corresponds to their distance from the diffuse surface. The reflection is most sharp and bright when an object is in direct contact with the surface.

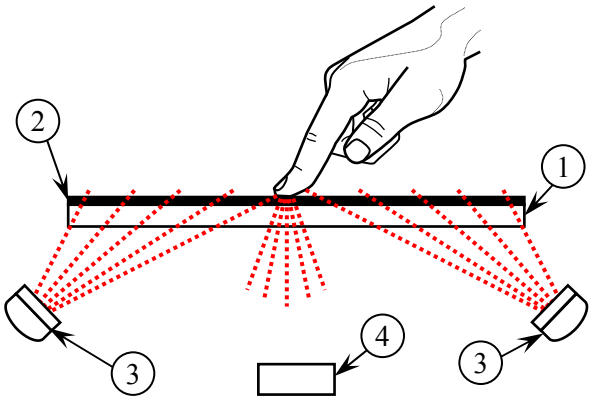


Figure 3: Diffuse Illumination: (1) glass or plexiglass, (2) diffusing layer, (3) IR illuminator, (4) IR camera

Advantages

- can recognize multiple touches as well as various shapes
- objects near to the surface can be recognized without direct contact with the surface
- can be used to recognize objects using special tags glued to their bottom
- good scalability
- unlike FTIR, DI does not require special acrylic glass and allows for adding another protection surface
- simple design

Disadvantages

- size (dimensions depends on the viewing angle of the camera)
- uniform infrared illumination of the surface is complex
- lesser contrast than FTIR
- can be affected by ambient infrared light

This method can be combined with the already mentioned FTIR method. In the way the accuracy of detection can be improved.

Diffused Surface Illumination (DSI)

Diffuse Surface Illumination method resolves the main issue of previously mentioned DI method – the uniformity of infrared illumination [4]. DSI requires special material of the projection surface. As shown in Figure 4, this material contains small particles that behave as microscopic mirrors that reflect infrared illumination from the sides uniformly out of the surface. Objects above the surface reflect this light back to the camera similarly as in the DI method.

Advantages

- similar as the already mentioned DI method
- uniform illumination is easier to achieve
- switch over to the FTIR method is simple (replacement of the surface glass)

Disadvantages

- size (dimensions depends on the viewing angle of the camera)
- requires a frame of the LEDs
- requires special glass *(Enlighten)*
- less contrast than DI (micro-mirrors reflects the light partially down to the camera)
- can be affected by ambient infrared light

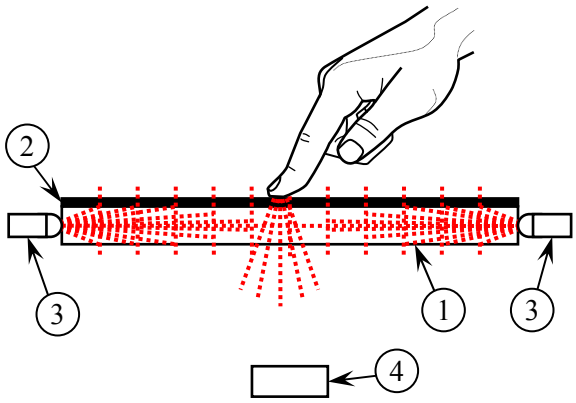


Figure 4: Diffused Surface Illumination: (1) *Endlighten* plexiglass, (2) diffusing layer, (3) IR LED, (4) IR camera

Matrix of Infrared Transceivers

Optical methods mentioned above require space behind the touch surface to accommodate the infrared camera. Size of the device depends on optical properties of this camera, in front of all on the viewing angle. This issue can be to some point resolved by using mirrors. If there is a demand for a flat device, matrix of infrared transceivers can be used. [4]

Principle of this method is following: Infrared emitters and sensors are uniformly placed out under the surface of touch panel (eventually LCD panel). Sensors detect light that is reflected from objects on or near the surface. This method has been used in system *ThinSight* [9] and in the new version of Microsoft Surface 2.0 [10].

Advantages

- thin form factor
- supports recognition of objects that are near to the surface (similarly to DI and DSI)

Disadvantages

- complex manufacturing process
- problematic scalability
- limited resolution

Computer Stereo Vision

Unlike already mentioned approaches, this method requires two cameras that capture the surface from two different places. Difference of these two images is used for computation of position of the objects (on or near the surface) [11]. The principle is similar as human stereoscopic vision.

Advantage of this method is that is supports recognition of objects (and gestures) that are more distant from the projection surface. However, this method requires complex calibration and processing and the surface has to be transparent.

2.2 Scanning Through the Screen

This section describes possibilities how to gain a visual input. Current technologies provide a simple way how to capture images of the user, even from different angles. To capture an image directly from the position of projection surface is more complex. The motivation is for example to enable users to keep the eye contact during a video-call or to capture images of objects that are near to the projection surface and therefore cannot be captured from its sides.

Currently there are more possibilities how to resolve this issue [12]. One of them is to use a semi-transparent mirror, but it has significant drawbacks. It requires additional space in front of the display for the mirror and it prevents access to the display surface. Moreover, prevention of unwanted reflections requires shielding.

Transparent display would be a better option. An LCD panel is not appropriate, because its transparency is only about 8% [13]. An OLED display is more promising, but it is currently too expensive, especially in sufficient dimensions [14]. Another option is to adapt rear projection to support this feature.

There is a market-available material – called *HoloScreen* or HOPS (Holographic-Optical Projection Screen) – which allows rear projection, while maintaining transparency [11]. It consists of holographic film, which redirects light depending on an angle of incidence [15, 16]. Problem is that the projected image is partly reflected back to the camera, so it interferes with the captured picture. Theoretically, it can be subtracted by the software, while the projected image is known. The reflection can be also decreased by using an antireflective film. The image mixing effect can be avoided by capturing in infrared light, if the application allows that. Last option is to use a shutter in front of the projector and closing it synchronously with camera shooting.

Another option is to use a projection surface that can be switched to a transparent mode [17]. This can be achieved for example by using a *Polymer Dispersed Liquid Crystal* (PD-LC) panel, which is used as a privacy glass. This material is diffusive translucent but not transparent in the initial state; therefore it can be used for the rear projection. It can be electrically switched to a transparent state. In this mode, pictures can be taken directly through the surface. The switching time is around 100 milliseconds. If there is a need for faster switching (e. g. for video capturing), an advanced technology called *Polymer Stabilized Cholesteric Textured Liquid Crystal* (PSCT-LC) can be used. This technology allows the state change in less than 0.5 milliseconds [18].

HoloScreen is used in *TouchLight* system [11] in combination with stereo-vision multi-touch technology. The switchable diffuser has been already used in several research works, usually with the PD-LC [17, 19]. The faster material has been introduced in *SecondLight* project [18]. The transparent state is not used for image capturing there (although the authors suggest that for the following work). In *SecondLight*, a second image is projecting through the screen onto another surface.

3 Our Solution: Interactive Table

This section describes our implementation of an interactive table. It summarizes our goals and main design steps in both hardware and software perspective.

3.1 Implementation Goals

In order to explore the mentioned methods of interaction, we decided to implement them in our own prototype. Our objective is to combine a multi-touch screen and the ability to take high-resolution photos through the projection plane in one device. Multi-touch displays become very popular these days. However the scanning feature is not so common in current implementations and looks promising from the interaction point of view (as have been discussed in chapter 1).

3.2 Hardware Design

The optical based technologies provide an easy way to construct a multi-touch screen in constraint (not industrial) conditions. These technologies are relatively cheap and moreover facilitate wider range of capabilities. We have chosen the technology of *rear diffused illumination* (DI), which can be easily combined with the intended scanning feature. Picture is projected by the projector from the rear on the diffusing plane.

The uniformity of IR illumination is achieved by using two *Infra Red LED Light Bars* provided by *Environmental Light* [20] and proper installation along both sides. These LEDs operates in 850 nm with high flux and wide beam angle. The capturing camera is equipped with IR-pass filter on same wavelength. Indirect ambient light does not strongly disturb the function thanks to the LED brightness, uniform illumination and a narrow band.

As mentioned, the ability of taking photos through the screen can be achieved in more ways. Technology with switchable diffuser (PD-LC) is simple, easy to construct, and does not require large form factor.

Placement of the main components is depicted on the Figure 5. The surface is illuminated by the two infra-red LED illuminators. The modified (IR-cut filter replaced by

IR-pass filter) PS3 Eye camera is capturing image in infra-red and the video-stream is transferred to the computer using USB. There is another high-resolution camera Cannon EOS 400D for taking photos (with IR-cut filter installed), the camera is also controlled via USB. This camera is located eccentrically (near the projector) to avoid reflections from the projector. For taking a photo, the diffuser is switched to the transparent state for a short time and the projector lightens the scanned area. The switching is achieved by using a remotely switchable socket, which is controlled from the computer over RS-232 [21].

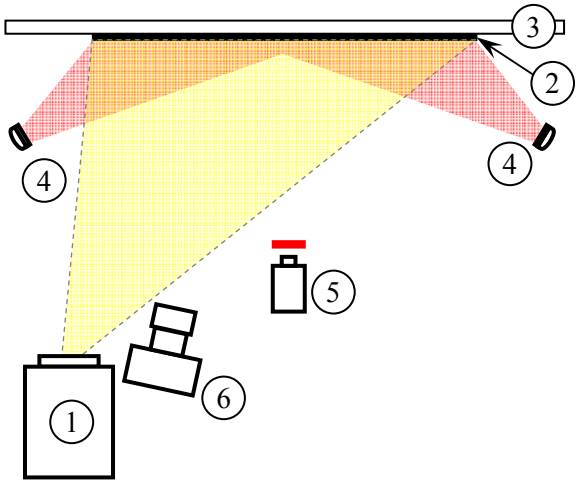


Figure 5: Prototype design: (1) projector, (2) switchable diffuser, (3) transparent plexiglass, (4) infrared illuminators (850 nm), (5) camera with IR pass filter (850 nm), (6) high-resolution photo camera

3.3 Software Design

After the multi-touch control became popular, several open-source implementations have been developed [22]. These solutions process video stream and calculate positions of touches. There are also frameworks and Software Development Kids (SDK) to simplify development of multi-touch applications. Some promising implementations are discussed in the following section.

3.3.1 Computer Vision

The first group of software solutions takes video-stream as an input, analyzes it and recognizes touches. The touch events are posted to the next application layer.

Community Core Vision (CCV) [23] is implementation by community *NUI Group*, under continuing development. It is intended for any camera-based optical technology. It can recognize fingers, and since version 1.4 also objects and tags. CCV is implemented in C++ for *Windows*, *Linux* and *MacOS*. Recognized touch events are sent to application layer through TUIO protocol [24] over UDP or as a native XML for *Adobe Flash* applications.

reacTIVison is simpler than CCV. It is developed in C++ by *Music Technology Group* at the *Universitat Pompeu Fabra* in Barcelona [25]. The TUIO protocol has been primarily designed for this software. Unlike CCV *reacTIVision* was focused for tags recognizing, but there is a support for fingers detection since version 1.4.

Bespoke Multi-Touch Framework is complex multi-touch framework implemented in C# by Paul Varcholik for *Windows* platform [26]. However, it is not further developed.

3.3.2 Application Frameworks

The next software layer processes the data about touches and provides support for developing multi-touch applications.

Project **MultiTouchVista** [27] developed by Daniel Danilin takes messages from TUIO protocol [24] (or inputs from multiple mice) and translated them to standard *Microsoft Windows* multi-touch event messages. It provides multi-touch driver for *Windows 7*, allowing utilize the embedded multi-touch support of this system *(Windows Touch)*. This allows using *Microsoft Surface Toolkit* and ensures the compatibility with other multi-touch devices including *Microsoft Surface 2.0*.

Microsoft Surface Toolkit [28] is a SDK for developing multi-touch applications for *Windows Touch* (native multi-touch interface in *Windows 7*). The toolkit contains controls, templates and application examples using application programming interface (API) of the *Windows Touch*.

PyMT is a *Python* module for developing *OpenGL* applications with multi-touch support [29]. It can be used on *Windows*, *Linux* or *MacOSX*, implements TUIO client and provides a large set of controls and tools usable in multi-touch applications. Several interesting examples are included.

3.3.3 Additional Software

Operating system **Microsoft Windows 7** was chosen because of its native support for multi-touch and compatibility with other layers and large set of similar devices. *Windows Presentation Foundation* framework version 4.0 contains tools for developing applications for *Windows Touch*.

Sony, the manufacturer of the **PS3 Eye** camera does not provide driver for personal computers. Fortunately, **CodeLabratories** developed one and provides it for free [30]. *CodeLabratories* also implements SDK allowing full control of the camera.

The high resolution photo camera used can be controlled via SDK, which is provided by *Canon* on explicit demand [31]. The current version of **Canon EOS Digital SDK (EDSDK 2.7)** has been used in our prototype.

The PD-LC controlling through the switching socket must be implemented. We have to ensure the right synchronization with **image capturing**. The .NET platform has been chosen as an implementation environment and C# as a programming language. Final implementation should be a dynamic library with a simple application interface (API).

3.3.4 Software Choice

Based on the previous analysis following software components has been chosen:

- operating system: **Microsoft Windows 7**
- camera driver: **CL Eye Platform Driver** [30]
- touch recognition: **CCV 1.4** [23]
- event handling: **CCV 1.4**, **MultiTouchVista** [27]
- switching controling: **our C# implementation**
- photo camera driver: **Canon SDK** [31]
- synchronization between taking photos and diffuser switching: **our C# implementation**
- additional application frameworks: **Microsoft Surface Toolkit** [28], **PyMT** [29]

Figures 6 and 7 describe links among these components.

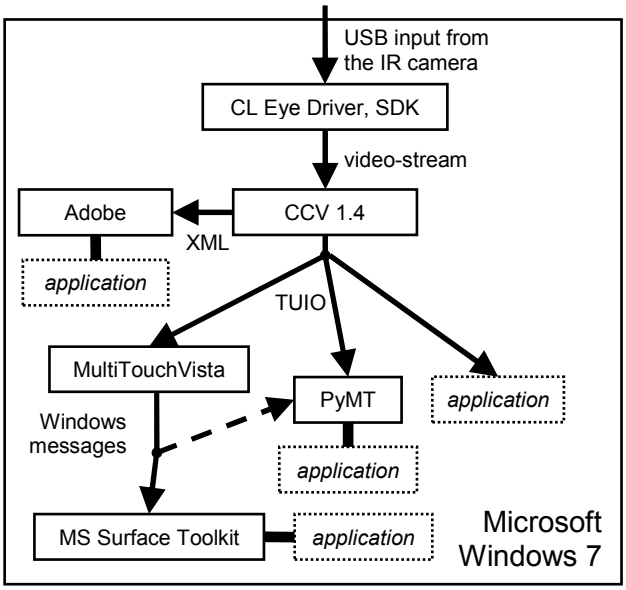


Figure 6: Scheme of the multi-touch software components

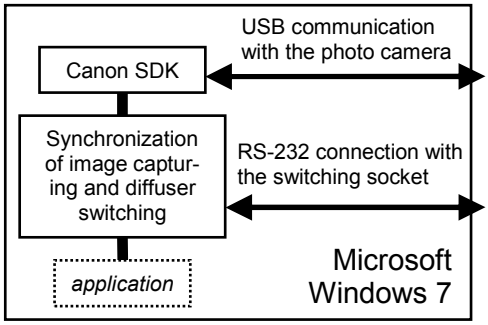


Figure 7: Scheme of the capturing software components

4 Results

This work has two main achievements. The first one is a summary of technologies for touch input and scanning from the screen direction. The second achievement is a functional prototype of a device which implements both multi-touch screen and image capturing through the display. The multi-touch input is achieved by using CCV as shown in Figure 8.

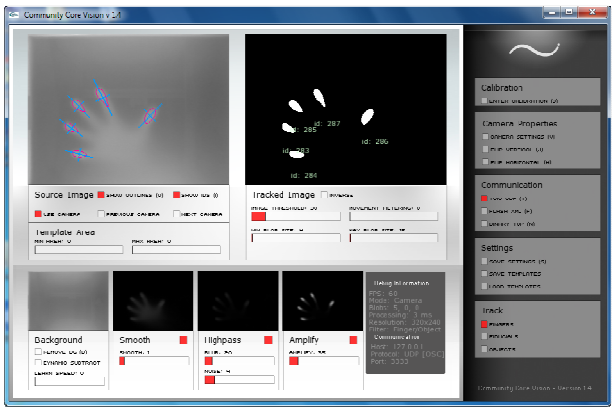


Figure 8: Touch recognizing by the Community Core Vision – version 1.4

Figure 9 shows the proposal and the final prototype.

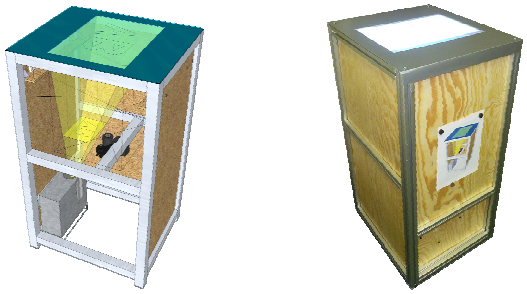


Figure 9: Our device: (left) sketch, (right) final prototype

5 Conclusions and Future Work

During the work it became clear that the issue of the touch input technologies is large. There are many approaches, the main are summarized in this paper. The aim of implementation has been fulfilled, and relatively cheap device was built.

The implemented device provides additional possibilities comparing to other multi-touch tables. Applications utilizing the scanning feature are subject of the immediate future work.

Future work should be focused on the software optimization and interaction design. Usability of implemented features should be evaluated. Additionally, barcode recognition can be added.

An automatic trimming of captured image would be very useful. In the current version, the whole display is lightened, while an image is captured. This is because the captured object should be well illuminated for a good picture. But the light can be shined directly to user's eyes, if no object occludes. If a good detection of object were implemented, the illumination could be focused only to the object.

There are also many possible improvements in the hardware implementation. The height of the device can be reduced by using front-surface mirrors and/or short-throw projector. Larger screen can be made (possibly with multiple cameras). Other form-factors of interactive systems (beside table) could be investigated.

Acknowledgments

This research has been done within project *Automatically Generated User Interfaces in Nomadic Applications* founded by grand no. SGS10/290/OHK3/3T/13 (FIS 10-802900) *(Studentská grantová soutěž)*.

References

[1] Buxton, B: *Multi-Touch Systems that I Have Known and Loved* [Online]. 23. 12. 2010. [Cited: 2010/28/12]. http://www.billbuxton.com/multitouchOverview.html

[2] Han, J. Y.: *Low-cost Multi-touch Sensing Through Frustrated Total Internal Reflection*. UIST '05 Proceedings of the 18th annual ACM symposium on User interface software and technology. Pages 115 – 118. ACM, New York, 2005. ISBN: 1-59593-271-2, DOI: 10.1145/ 1095034.1095054.

[3] Vekobs: *Technologie dotykových panelů* [Online]. Vekobs Webpage. [Cited: 2010/02/05]. http://www.vekobs.cz/cz_technologie.htm

[4] Schöning, J.; Brandl, P.; Daiber, F.; Echtler, F.; Hilliges, O.; Hook, J.; Löchtefeld, M.; Motamedi, N.; Muller, L.; Olivier, P.; Roth, T.; Zadow, U.: *Multi-Touch Surfaces: A Technical Guide*. Technical Report TUM-I0833. Technical University of Munich. Munich, 2008.

[5] NUI Group: *Natural User Interface Group* [Online]. [Cited: 2010/12/01]. http://nuigroup.com

[6] NUI Group: *Laser Light Plane Illumination (LLP)* [Online]. NUI Group Community Wiki. 2009/05/11. [Cited: 2010/12/28]. http://wiki.nuigroup.com/Laser_Light_Plane_Illumination_(LLP)

[7] Han, J. Y.: *Multi-touch sensing through frustrated total internal reflection*. US Patent Application 20080179507. Submitted: 2007/08/93. Published: 2008/07/31. New York, USA.

[8] NUI Group: *What is a compliant surface?* [Online]. NUI Group Community Wiki. 2009/10/29. [Cited: 2010/12/28]. http://wiki.nuigroup.com/What_is_a_compliant_surface%3F

[9] Hodges, S.; Izadi, S.; Butler, A.; Rrustemi, A.; Buxton, B.: *ThinSight: versatile multi-touch sensing for thin form-factor displays*. Proceedings of the 20th annual ACM symposium on User interface software and technology. Pages 259 – 268. ACM, New York, 2007. ISBN: 978-1-59593-679-0; DOI: 10.1145/1294211. 1294258.

[10] Oiaga, M.: *Microsoft Surface 2.0 Features Windows 7 and New PixelSense Technology* [Online]. Softpedia. [Cited: 2011/01/31]. http://news.softpedia.com/news/ Microsoft-Surface-2-0-Features-Windows-7-and-New-PixelSense-Technology-176602.shtml

[11] Wilson, A. D.: *TouchLight: An Imaging Touch Screen and Display for Gesture-based Interaction*. ICMI '04 Proceedings of the 6th international conference on Multimodal interfaces. Pages 69 – 76. ACM, New York, 2004. ISBN: 1-58113-995-0, DOI: 10.1145/1027933.1027946.

[12] Kollarits, R.; Woodworth, C.; Ribera, J.; Gitlin, R. D.: *An Eye-Contact System for Videophone Applications Using a Conventional Direct-View LCD*. Digest of technical papers of Society for Information Display International Symposium: SID1995. Str. 765 – 768. SID, 1995.

[13] Seetzen, H.; Heidrich, W.; Stuerzlinger, W.; Ward, G.; Whitehead, L.: *High dynamic range display systems*. ACM Transactions on Graphics (TOG) – Proceedings of ACM SIGGRAPH 2004. Volume 23, Issue 3. Str. 760 – 768. ACM, New York, srpen 2004. ISSN: 0730-0301, EISSN: 1557-7368, DOI: 10.1145/1015706.1015797.

[14] Goble, G.: *Is OLED Already Dead?* [Online]. Digital-Trends. 2010/04/07. [Cited: 2010/12/28]. http://www.digitaltrends.com/features/is-oled-already-dead/

[15] sax3d.com Chemnitz: *sax3D Holographic Systems* [Online]. [Cited: 2011/01/01]. http://www.sax3d.com/en

[16] Kuechler, M.; Kunz, A.: *HoloPort – A Device for Simultaneous Video and Data Conferencing Featuring Gaze Awareness*. Virtual Reality Conference. Pages 81 – 88. IEEE Computer Society, Alexandria, March 2006. ISBN: 1-4244-0224-7, DOI: 10.1109/VR. 2006.71.

[17] Shiwa, S.; Ishibashi, M.: *A Large-Screen Visual Telecommunication Device Enabling Eye Contact*. Digest of technical papers of Society for Information Display International Symposium: SID1991. Str. 327 – 328. SID. 1991.

[18] Izadi, S.; Hodges, S.; Taylor, S.; Rosenfeld, D.; Villar, N.; Butler, A.; Westhues, J.: *Going Beyond the Display: A Surface Technology with an Electronically Switchable Diffuser*. UIST '08 Proceedings of the 21st annual ACM symposium on User interface software and technology. Str. 269 – 278. ACM. New York, 2008. ISBN: 978-1-59593-975-3, DOI: 10.1145/1449715.1449760.

[19] Gross, M.; Würmlin, S.; Naef, M.; Lamboray, E.; Spagno, C.; Kunz, A.; Koller-Meier, E.; Svoboda, T.; Van Gool, L.; Lang, S.; Strehlke, K.; Moere, A. V.; Staadt, O.: *blue-c: a spatially immersive display and 3D video portal for telepresence*. ACM Transactions on Graphics (TOG) – Proceedings of ACM SIGGRAPH 2003. Volume 22, Issue 3. Str. 819 – 827. ACM. New York, June 2003. ISSN: 0730-0301, DOI: 10.1145/ 882262.882350.

[20] Environmental Lights: *Multi Touch Screen Pair Kit* [Online]. [Cited: 2010/10/10]. http://www.environmentallights.com/products/13093/LED_ Bar_12_inch_850nm_infrared_pair_kit_Euro

[21] Mikrovlny: *RS232 Watchdog* [Online]. Mikrovlny, s.r.o. [Cited: 2010/10/16]. http://www.mikrovlny.cz/cz/produkt/16

[22] NUI Group: *Applications and libraries* [Online]. NUI Group Community Wiki. 2010/06/16. [Cited: 2010/11/15]. http://wiki.nuigroup.com/Applications_and_libraries

[23] NUI Group: *Community Core Vision* [Online]. [Cited: 2010/12/29]. http://ccv.nuigroup.com/

[24] TUIO.org: *TUIO* [Online]. [Cited: 2010/12/29]. http://www.tuio.org/

[25] Music Technology Group: *reacTIVision* [Online]. [Cited: 2010/12/29]. http://reactivision.sourceforge.net/

[26] Varcholik, P.: *Bespoke Multi-Touch Framework* [Online]. [Cited: 2010/12/29]. http://www.bespokesoftware.org/wordpress/?page_id=41

[27] Danilin, D.: *Multi-Touch Vista* [Online]. CodePlex. 2010/11/07. [Cited: 2010/12/29]. http://multitouchvista.codeplex.com/

[28] Microsoft: *Surface Toolkit for Windows Touch Beta* [Online]. MSDN Library. [Cited: 2011/01/02]. http://msdn.microsoft.com/en-us/library/ee957351.aspx

[29] PyMT: *PyMT: Open source library for multitouch development* [Online]. [Cited: 2010/12/29]. http://pymt.eu/

[30] Code Laboratories: *CL Eye Platform Driver* [Online]. Code Laboratories. [Cited: 2010/11/03]. http://codelaboratories.com/products/eye/driver/

[31] Canon: *Digital Imaging Developer Programme* [Online]. Canon Europa. [Cited: 2010/12/29] http://www.didp.canon-europa.com/

Overview of current developments in haptic APIs

Petr Kadleček
Supervised by: Petr Kmoch

Charles University in Prague
Faculty of Mathematics and Physics
Prague / Czech Republic

Abstract

Haptic technology as a key part of human-computer interaction allows us to use sense of touch in virtual reality by kinesthetic feel using force feedback. Increased production of haptic devices in recent years supported the development of many tools and libraries for programming applications with support of haptics. This paper introduces haptic technology and focuses on comparison of haptic application programming interfaces, especially on open-source and cross-platform solutions. We present different types of abstraction layers used in haptic APIs, basic haptic rendering methods and effects as well as a general overview of design concepts used in selected APIs. CHAI 3D haptic library is analyzed in more detail.

Keywords: haptic technology, human-computer interaction, haptic rendering, CHAI 3D, H3D API

1 Introduction

Kinesthetic sense provides us with information about movement and position of our body parts in the environment. We are able to feel various forces in different directions and use this information to determine the size, shape and other characteristics of objects we touch and forces they exert. Haptic modality of human-computer interaction utilizes sense of touch which is generally incorporating hands, upper torso, head and other parts of the body. The purpose of a haptic device is to generate force feedback of a given direction and magnitude in a specified workspace and send the position of a control part of the apparatus to the computer. One of the most valuable applications of haptic devices is in medicine (simulations of surgical operations, teleoperation, virtual palpation [5]). Haptic devices are also valued as assistive technology for visually impaired or blind people [7]. Other applications can be found in military, painting, CAD systems and gaming.

Haptic devices can be generally divided by the dimension of an orientation ability called degrees of freedom (DOF). That is basically translation (3-DOF) and translation combined with rotation (6-DOF). A typical example is a movable grip for 3-DOF devices (e.g. Novint Falcon shown in Figure 1) and a pen on a pivot with the ability to rotate and translate in all three dimensions for 6-DOF devices. There are also 6/3-DOF devices that combine 6-DOF positioning and 3-DOF force feedback (e.g. PHANToM Desktop shown in Figure 1). 7-DOF devices have a scissors snap-on, a thumb-pad or any other extra grip.

Figure 1: PHANToM Desktop (on the left) and Novint Falcon (on the right)

Common comparable properties which can be found in technical specifications of haptic devices include: *workspace* specifying a maximal reach of a touch tool (often measured in inches) and maximal rotation abilities if appropriate, *position resolution* of a touch tool measured in dots per inch (DPI), *maximal force* specified in newton unit or as a force capability in kilograms or pounds and *stiffness* of a haptic device along a degree of freedom measured in newtons per metre.

While force feedback gives a sense of force or generally a kinesthetic feel, tactile and touch sensing is used when one wants to feel pressure, heat or fine textures (and any other sensation felt by the skin). Haptic devices do not usually provide cutaneous sensation and it should not be confused with kinesthetic feel. Technology prototypes using both kinesthetic and tactile feedback have been proposed [6].

Although the sense of touch is not as acute as hearing, its accuracy is somewhere in between sight and hearing. Humans need approximately 1000 Hz frequency of haptic feedback to achieve smooth force perception. If the frequency is smaller than 1 kHz, a haptic stimulus felt by organs of human kinesthesia is unrealistic and may even lead

to system instability, potentially causing injury or damaging the device as stated in [11]. This means that a haptic loop has to be at least 30 times faster compared to minimal real-time computer graphics rendering rates, which demands great optimizations in haptic applications.

The remainder of this paper is organized as follows: In Section 2, we discuss different abstraction layers of haptic APIs. Basics of haptic rendering algorithms, haptic effects and other extensions are presented in Section 3. Sections 4, 5 and 6 are devoted to overview of CHAI 3D, HAPI, H3DAPI and other haptic APIs. The paper concludes with a table of haptic APIs specification and benchmark.

2 Abstraction layers of Haptic APIs

There are various methods of implementing haptic device control into an application ranging from the lowest driver layer to the highest scene graph layer. The most important decision a software architect has to make is a choice of the particular abstraction layer (shown in Figure 2) at which the rest of the application communicates with haptics.

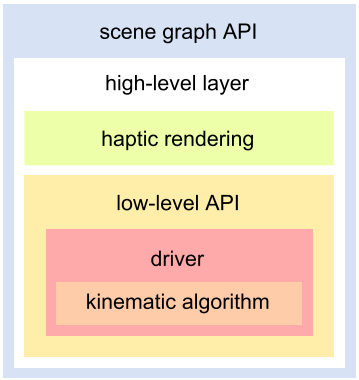


Figure 2: Abstraction layers of haptic APIs

2.1 Driver layer and kinematic algorithm

The lowest layer at which the programmer can communicate with the device is a driver of the operating system. At this layer the driver receives raw data through a serial bus (e.g. USB, IEEE 1394) from encoders that has to be processed with kinematics algorithms to get the data that corresponds to a three-dimensional vector of the haptic tool position in Cartesian coordinates. Kinematic algorithms are often a part of the driver because of specific technical specifications of every device. Manual initialization, opening and closing communication with the device or an inverse kinematics algorithm which computes force data in the application and sends it to the device to compute angles at haptic device joints is also essential. To preserve a smooth haptic response thread handling has to be done. For this reason, an extra haptic thread which calculates physics in the application is necessary.

The driver layer provides the fastest and the most precise response but demands a great effort to get the device working. Support of any other haptic device that has no compatible communication protocol means rewriting a lot of source code.

Manufacturers of haptic devices often provide optimized and well documented drivers in the C or C++ programming language. There are also open source and cross platform drivers that can provide support in officially unsupported operating systems such as Linux or Mac OS.

2.2 Low-level API

While the driver layer communicates in raw data, a low-level API hides the kinematics algorithm implementation from the programmer and allows developers to work directly with position, rotation and force vectors in the application. Many low-level APIs works as a common interface for different drivers which is very helpful when supporting a lot of haptic devices. A device handler is then used for getting information on haptic devices available on the current machine. Reading information from haptic devices may be blocking or non-blocking. Blocking servo loop callback stops the application thread where the function was called and reads the data at the frequency of haptic interface servo loop. A typical haptic application using low-level API is presented in algorithm 1.

Algorithm 1 Application using low-level haptic API

```
Initialize haptic device handler
number of haptic devices ← haptic device handler
if number of haptic devices = 0 then
    Exit
end if
specification of haptic devices ← haptic device handler
Initialize specified haptic device(s)
while Simulation running do
    position ← haptic device {blocking or non-blocking}
    compute force {haptic rendering}
    force → haptic device
end while
```

2.3 High-level layer

A graphical and haptical data representation of a model may be very similar or sometimes even identical. Integration of graphics and haptics into one API is therefore reasonable. There are several different approaches to create high-level API. One of the most intuitive way of incorporating haptics into application is based on calling similar functions that are provided in OpenGL graphics library.

A layer which handles computation of forces for a given model is called a haptic rendering layer. We describe it in more detail in the next section.

2.4 Scene graph API

A scene graph haptic API often uses a tree structure of objects in the virtual world with a specific root node such as a world node. It is possible to apply graphical and haptical properties to an object and set the specific property recursively to its child objects.

A scene graph API often includes low-level APIs for haptics, graphics, physics and audio processing. It provides all the features of low-level APIs and even more by combining them together. Haptic and graphic rendering is essential in the scene graph API oriented on haptics.

The concept of combining low-level APIs into one often creates many drawbacks which the high-level scene graph API implementation may or may not hide from the programmer. Difficulties connected with such a combination of different APIs may result in a thorough problem analysis that may not even be solvable with a feasible effort because the API itself may be proprietary and authors may not support the API any more.

A scene graph haptic API is the best choice for prototyping an application when the speed of development is crucial and performance is not a priority. Support of a scripting language or standard file format representation of a scene helps even more with rapid development.

3 Haptic rendering

One of the most important algorithmic problems associated with haptics is computation of interactions between the haptic tool and virtual objects. Creating a convincing force reaction on a complex object is a nontrivial task that is dependent on data representation. The technique of haptic interaction processing in the virtual scene is called haptic rendering (or haptic display). As in graphic rendering, where the image is composed from a model based on a virtual camera position, the process of haptic rendering returns a force on the basis of a model with which the haptic tool interacts. Creating a good haptic rendering algorithm is a struggle to maintain realistic force feedback without using cumbersome computations which raise memory and CPU requirements.

There are basically two accepted standard methods that are implemented in high-level haptic APIs for 3-DOF haptic rendering: God-object method by Zilles et al. [15] and Virtual proxy method by Ruspini et al. [11]. It should be noted that even though there are many articles concerning 6-DOF haptic rendering, there is no standard widely-used implementation.

The maximal stiffness capability along any degree of freedom is limited on every haptic device. Therefore, a user may move a haptic tool with a force which lets them penetrate into a rigid body or any kind of object. Haptic rendering algorithms are trying to solve this problem by exerting an adequate force that is pushing a haptic tool away from the object.

3.1 Penalty based methods

The simplest type of haptic rendering technique specifies a force vector for every point in a scene by calculating the nearest resting position of a haptic tool also represented as a point. If the haptic interface point is outside the object, the resulting force is zero, otherwise the force vector has a magnitude proportional to the penetration distance. This kind of method is also called *vector field method* [15] or *penalty based method* [11]. This technique, however, has many drawbacks which make it useless for at least plausible simulations. As this method does not save a history of haptic interface point movement, discrete space of haptic servo updates may result in unnoticed penetration through an object in one haptic loop step as shown in Figure 3 on the left. Another pop-through problem may come up when penetration is too deep and the desired nearest resting point is on the other side of the object as shown in Figure 3 on the right.

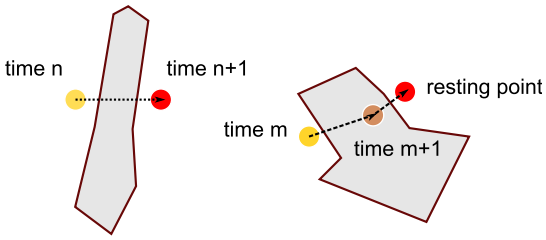


Figure 3: Pop-through problems with penalty based methods

3.2 God-object method

To solve pop-through problems mentioned in penalty based methods a God-object method was proposed [15]. The God-object represents a virtual point in the scene that is not able to penetrate into rigid bodies and thus behaves correctly. A position of the God-object is updated in every haptic loop step.

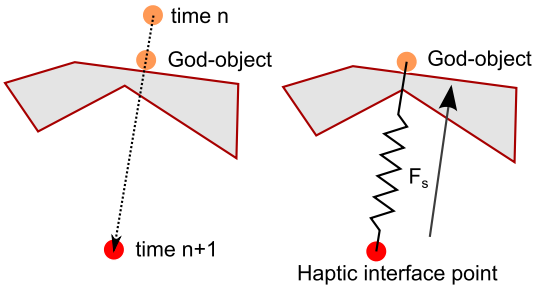


Figure 4: God-object

If the haptic interface point (HIP) penetrates into an object, the movement of god-object towards HIP is constrained by a surface of this object and the resulting force is calculated by simulating an ideal mass-less spring (shown

in Figure 4) which is, according to Hooke's law, defined as follows:

$$F_s = -k\Delta x = -k(x_{\text{HIP}} - x_{\text{GodObject}}) \quad (1)$$

where Δx is a displacement of spring and k is a spring constant defining the stiffness of the surface.

The God-object method can be easily extended [14] to support static and dynamic friction on rigid bodies which is essential to achieve realistic haptic stimulus. Haptic shading, an analogous algorithm to Phong shading can be applied on force feedback on surface normals to create an effect of smooth surface. Another association to computer graphics is in the use of textures. A haptic texture mapped on the object can be used to simulate different kinds of materials such as wood, stone or metal. Realistic haptic texture rendering has been investigated in [3].

3.3 Virtual proxy method

Polygonal meshes often contain small surface gaps because of low-quality digitization or non-precise modeling. When the god-object enters a mesh through a small gap, the user gets stuck inside the mesh until he finds the gap again. To resolve the problem we either fill in small gaps in the process of loading the mesh or we set a radius of the god-object in collision detection with constraint planes. The Virtual proxy method [11] proposes to treat a presentation of the haptic tool in the virtual environment as a sphere (as shown in Figure 5). Extensions discussed in God-object method are applied simply by moving the proxy and thus changing the resulting force.

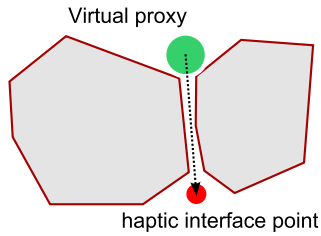


Figure 5: Virtual Proxy

In the remainder of this paper, we will examine several common haptic APIs in more detail.

4 CHAI 3D

CHAI 3D [4] is a scene graph API written in the C++ programming language with aim to create a modular, open source and cross platform haptic API with a wide support of different haptic devices (and a virtual device working on Microsoft Windows platform). CHAI 3D is licensed under GNU General Public License (GPL) version 2 but also offers a Professional Edition License. The main reason to create CHAI 3D was that all available APIs developed by manufacturers of haptic devices were proprietary and supported only the one specific device or a group of devices from the manufacturer.

The scene graph capabilities of CHAI 3D mainly focus on haptics combined with graphics. It does not include any extra visual or sound effects but it does propose lightweight and compact functionality. CHAI 3D is definitely not the API with tons of functions ready for the implementation of sophisticated applications. It is rather the API for academic and research use where the extra functionality can be easily added.

Though the API manual or tutorials do not yet exist, the source code is very well documented and is very easy to read and scan through. The reference guide generated by a Doxygen documentation system could serve as a quick guide over the source code but it is not a comprehensive source of learning CHAI 3D. Authors of CHAI 3D recommend to learn by the examples in packages for different platforms. This method gives the learner a decent overview of the API but does not allow to fully understand some fundamental characteristics of the API which makes the learner read part of the API source code eventually.

4.1 Low-level use of API

Though the CHAI 3D library is a scene graph API, use of CHAI 3D as a low-level communication layer is convenient. CHAI 3D provides support of many devices and an easy to use device handler *cHapticDeviceHandler*. Every device is then treated as a generic haptic device *cGenericHapticDevice* with basic ability to get a position, set a force, device communication opening, initialization and closing.

4.2 CHAI 3D scene graph

A scene graph of CHAI 3D contains standard shapes, meshes, virtual cameras and lights. The main unit of all objects in the scene graph is a *cGenericObject* class which inherits from a general abstract type *cGenericType*. The generic object creates a tree structure of objects using a standard template vector class of children objects in a *m_children* member. All methods for object modification or property setting allow propagation to children by setting an optional function parameter *a_affectChildren*, which is by default set to false. CHAI 3D scene graph has one root node class for every object in the scene called *cWorld*. This class is essential for further communication with graphics and haptics.

The API contains only three standard object shapes (two implicit surface objects): sphere (*cShapeSphere*) defined by a radius, torus (*cShapeTorus*) defined by an inside and an outside radius and line (*cShapeLine*). Beside standard shapes implemented in CHAI 3D API, it is possible to load complex meshes in OBJ and 3DS formats.

4.3 Haptic tool

The scene graph representation of a haptic device is called a tool. An abstract class defining all tools in the scene graph is *cGenericTool*. The only specific tool that CHAI 3D provides at this time is a 3-DOF tool identified as a *cGeneric3dofPointer*. 6-DOF force rendering algorithms are not supported.

The generic tool is also a generic object which means that the tool has its position, rotation and all other object properties. The tool itself needs only a pointer to the haptic device from a device handler. It manages all the initialization automatically by calling a *start* method. A *stop* method does the opposite.

The default device mesh of the generic 3-DOF pointer displays the tool as a sphere. God-object algorithm with variable radius is used for the haptic force rendering for which there are two meshes representing the tool:

- a device mesh (*m_deviceMesh*) which represents the real current position of the haptic device touch tool
- a proxy mesh (*m_proxyMesh*) which represents a model of the haptic interface in the virtual environment

The force model is also defined as the abstract model (with a generic class *cGenericPointForceAlgo*) split into *cProxyPointForceAlgo* and *cPotentialFieldForceAlgo* classes. The *cProxyPointForceAlgo* class implements the God-object method and *cPotentialFieldForceAlgo* class process local interaction relating to haptic effects.

An overall force contains assigned local haptic effects and interaction forces computed on the base of haptic device properties (e.g. stiffness), a position relative to an interaction projected point on the interacting object surface and a best new position of the proxy model in the proxy point force algorithm. Interaction detection is not always precise especially in complex meshes and the proxy model gets sometimes stuck and generates excessive force.

The tool works in a workspace set by a radius. It is possible to change the radius and position of the workspace and its rotation relative to the scene. The tool is often attached to the camera so that the workspace corresponds to the view of the camera. A schema of haptic tool interaction process is shown in Figure 6.

4.4 Haptic effects

The CHAI 3D scene graph provides a set of haptic effects that can be assigned to implicit surface objects [12]. These effects are computed using a local interaction *computeLocalInteraction* method of each object. The mesh or any other complex object without overridden *computeLocalInteraction* method is not able to apply haptic effects because there is no way how to compute an interaction projected point from a generic object algorithm. Only the proxy point algorithm is used for these objects to calculate forces.

Haptic effects with the base abstract class *cGenericEffect* in the API are as follows:

- Magnetic model effect *cEffectMagnet* provides a magnetic field effect near the object
- Stick-slip effect *cEffectStickSlip* provides an effect of sliding one object on another with sticking caused by friction (e.g. rubber on a desk)
- Surface effect *cEffectSurface* provides a basic surface effect of a tool pushing against the object
- Vibrations effect *cEffectVibration* provides an effect of a vibration with a specific frequency and amplitude
- Viscosity effect *cEffectViscosity* provides an effect of a tool moving through a fluid

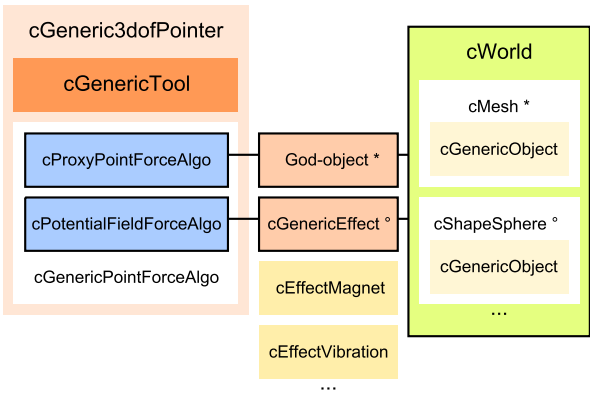


Figure 6: Schema of a haptic tool interaction process in CHAI 3D - effects can be applied only on implicit surface objects, God-object method is used for mesh objects in cWorld, as denoted by asterisk and circle

All effects are very sensitive to a good setting of properties such as a maximal stiffness of the haptic device. A relatively small change of effect properties can make a great difference in the effect perception and sometimes even a different driver may result in a different effect behavior.

4.5 ODE module

The CHAI 3D library does not implement its own rigid body dynamics simulation. There is, however, a module that connects the CHAI 3D scene graph with the Open Dynamics Engine (ODE) library.

Communication of CHAI 3D and ODE is handled by *cODE*, *cODEWorld* and *cODEGenericBody* classes. The API contains precompiled ODE libraries for both dynamic and static linking with double precision. Preprocessors definitions need to be set correctly in order to run an application properly without runtime errors.

Every object in the ODE simulation has to be added to a specific ODE world. Such an object is defined as an ODE generic body with properties of physical simulation and a CHAI 3D body image model of the scene graph. The ODE world is a generic object which behaves as a child object in the standard parent world object but has a list of *bodies* instead of a list of *children*. However, all recursive algorithms in CHAI 3D look up *children* list in the scene graph. For instance, it is therefore not possible to assign a haptic effect to an object in the ODE simulation because the rendering algorithm is using the mentioned recursion through children list. A fix of this behavior can be found in [10].

The ODE module enables creation of a dynamic box, sphere, capsule and a mesh from an assigned CHAI 3D body image model. Static planes are also available. A global gravity can be set as a three-dimensional vector describing a force. Calling an ODE world *updateDynamics* method with a step time function parameter updates the simulation. Though the implementation of dynamics into the scene graph is simple, a programmer still has to work with the ODE world as a separate world and encounters a lot of disadvantages when using recursive scene graph algorithms.

4.6 GEL module

The haptic technology utilizes an implementation of a deformable body simulation more than any other technology. CHAI 3D provides a module to create such deformable objects in the scene graph which uses the GEL dynamics engine developed at Stanford University.

As in the ODE module, the GEL module is implemented as a separate world (*cGELWorld*) of deformable objects. The main idea behind the deformation is a skeleton model made of nodes (*cGELSkeletonNode*) and links (*cGELSkeletonLink*) between them. Nodes are represented as spheres with a given radius and mass connected with elastic links with spring physics defined by elongation, flexion and torsion properties (as shown in Figure 7). Every node has its physical properties (linear damping, angular damping, gravity field definition) and provides methods to control force and torque.

The GEL module provides a simple way to add deformable objects to the scene graph, but integration of the GEL dynamics engine in the lower layer of the scene graph with automated skeleton modeling would considerably enhance the high level use of CHAI 3D.

5 H3D API and HAPI

H3D API [1] is a high level scene graph API developed by SenseGraphics. H3D API uses HAPI as a low-level layer for haptics, OpenGL for graphics and the X3D XML-based file format to represent the scene. The library is written in the C++ programming language and is licensed under GNU GPL v2. Closed source license for commercial use is also available.

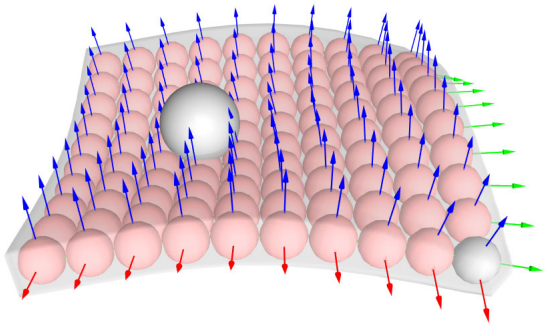


Figure 7: CHAI 3D GEL module example

5.1 X3D

The most interesting feature H3D API provides is scene definition in X3D file format. The whole scene with a camera set, lights, primitive objects, complex meshes, textures, etc. is defined as XML nodes. As X3D is originally web-based technology, a texture or any other object loaded from a file can have a URL path.

The haptic device is defined through a *DeviceInfo* node with the haptic renderer specification, position calibration and the proxy model appearance. H3D API implements all HAPI haptic rendering functionality to the X3D specification. For instance, to add a frictional surface effect to the shape in the scene, a XML node *FrictionalSurface* is added to the appearance node of the shape with appropriate properties.

H3D API also supports X3D routes which makes it possible to read data from one source and route it to a specified destination. That is for instance routing the position of the mouse from the *MouseSensor* node to the shape node position. A *PythonScript* node allows to route data from X3D to Python programming language functions.

5.2 Python interface

H3D API propose a very unique way of haptic programming using Python scripts on top of the X3D scene definition. A Python interface to the H3D API implements X3D creation and write functions, special bindable node access (haptic device info, viewpoint, etc.) and X3D field types so that it is possible to create a comprehensive application just using the X3D and Python when there is no reason to develop efficient real-time application.

5.3 Scene graph and C++

H3D API is not only the Python and X3D. The entire application can be written in the C++ programming language

for better performance. The C++ code allows to parse X3D strings which makes it easier to create objects or set materials in C++. This method should be used only in initialization of the scene because real-time X3D parsing in a graphics loop of the application would lower the performance.

H3D API is a perfect tool to create fast prototypes of applications using haptics. Python and X3D is available for a very rapid development and C++ for higher performance applications.

5.4 HAPI

HAPI [1] is a new complex open source high-level haptic API also developed by SenseGraphics licensed under GNU GPL v2. As with the H3D API, a closed source license is also available. HAPI is written in the C++ programming language and works on all major operating systems: Microsoft Windows, Linux and Mac OS.

HAPI is one of the most active haptic APIs supporting devices from Sensable, Force Dimension, Novint and Moog FCS Robotics. There are four haptic rendering algorithms available: God-object algorithm (described in Section 3), Ruspini algorithm - Virtual proxy method, CHAI 3D rendering, OpenHaptics rendering.

HAPI provides not only the basic device handling, but there is also a number of haptic force effects, surface effects, collision detection, primitive shape creation and thread handling. A very specific functionality is graphics rendering based shape creation. It allows a programmer to create haptic shapes using standard OpenGL drawing functions. A *FeedbackBufferCollector* class collects all triangles that are rendered via the OpenGL library.

HAPI is very well documented with an accompanying manual, reference manual generated by Doxygen documentation system and a lot of examples of all features. The source code of a basic device handling application written in HAPI using the *AnyHapticsDevice* class has just about 20 lines. HAPI can be downloaded as a Windows Installer or as the source code.

The manual and examples make HAPI very easy to use. HAPI is one of the best choice of commercial and non-commercial high-level APIs with a very good support from authors and can be also used as a low-level API.

6 Other haptic APIs

OpenHaptics[13] is a commercial software development toolkit designed for SensAble devices. The toolkit contains scene graph API for rapid development, high-level and low-level APIs and support for integration of haptics into existing applications. OpenHaptics is also available in Academic Edition.

There are many low-level APIs designed for specific devices: HDAL [8] (Novint Haptic Device Abstraction Layer) which is a commercial closed source SDK

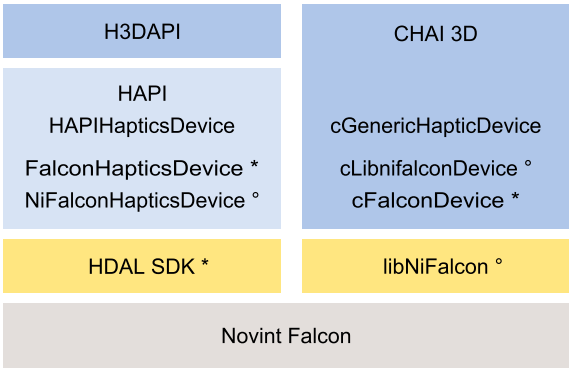


Figure 8: Haptic API abstraction layers for Novint Falcon. HDAL SDK wrapping classes are denoted by asterisk, libNiFalcon wrapping classes are denoted by circle

for Novint Falcon device working only on Microsoft Windows, libNiFalcon [9] - an open-source driver for Novint Falcon working on all major platforms or JTouch-Toolkit [2] (HDAL SDK and OpenHaptics HDAPI/HLAPI wrapper for Java platform). Example of haptic API abstract layers for Novint Falcon device is shown in Figure 8 (experimental implementation of libNiFalcon into CHAI 3D is a part of [10]).

Conclusion

We have introduced haptic technology and discussed aspects of programming with haptics. We have shown that there are many ways how to add support of haptic technology into an application using different abstraction layer of haptic APIs varying from haptic device driver, low-level APIs to high-level scene graph APIs. We have presented basic methods of 3-DOF haptic rendering - specifically the God-object method and Virtual proxy method which are used in high-level APIs such as CHAI 3D, HAPI or OpenHaptics. Relevant parts of CHAI 3D haptic library have been analyzed in detail including haptic tool, haptic effects or ODE and GEL module support. Very active haptic APIs HAPI and H3D API have also been analyzed. H3D API brings a possibility to create haptic applications in declarative programming language X3D with an interface to Python programming language. Another commercial device specific haptic APIs were mentioned such as HDAL SDK or OpenHaptics.

Final comparison of haptic APIs is given in Table 1. Computational load benchmark has been implemented in selected low-level haptic APIs as a simple force field haptic rendering algorithm. Benchmark ran on Intel Atom 330 1.6 GHz dual core CPU and results are shown in figure 9. Some APIs do not support blocking servo calls which make fast haptic rendering algorithms (under 1ms) to create pointless CPU load by polling data in loop. This

API	CHAI3D	HDAL SDK	JTouchTool.	libNiFalcon	HAPI	H3D API	OpenHaptics
Open source	●	○	●	●	●	●	○
Cross platform	●	○	*	●	●	●	●
License	GPL/C	C/N	GPL	BSD	GPL/C	GPL/C	C/A
Development state	●●○	●●●	●○○	●●○	●●●	●●●	●●●
API manual	○	●	○	○	●	●	●
API reference	●	●	●	●	●	●	●
Device range	●●●	●○○	●●○	●○○	●●●	●●●	●●○
Abstraction layer	High/Low	Low	Low	Low/Driver	High/Low	High	High/Low

Table 1: Haptic APIs comparison, C = commercial, N = non-commercial, A = academic, * = partial

behavior is taken into account and some APIs were extra benchmarked with simple polling prevention.

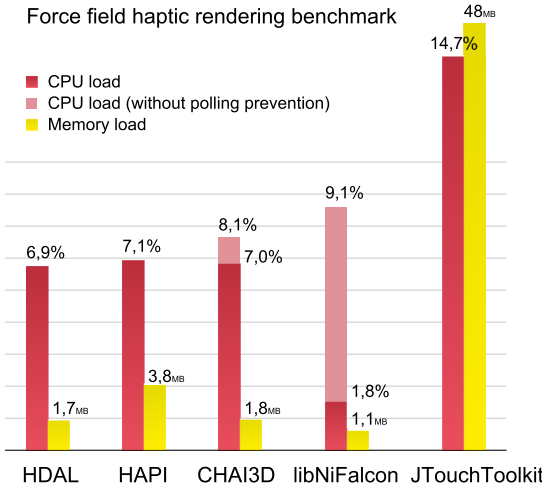


Figure 9: Benchmark based on simple haptic rendering algorithm simulating force field with ideal spring defined by Hooke's law

Acknowledgements

I would like to thank Petr Kmoch for his support and advice throughout the creation of this work.

References

[1] SenseGraphics AB. *H3D API - haptics software development platform*, 2011. `http://www.h3dapi.org/`.

[2] John Archer. *JTouchToolkit*, 2008. `https://jtouchtoolkit.dev.java.net/`.

[3] S. Choi and H.Z. Tan. Toward realistic haptic rendering of surface textures. In *ACM SIGGRAPH 2005 Courses*. ACM, 2005.

[4] Conti Francois et al. *CHAI 3D set of libraries*, 2009. `http://www.chai3d.org/`.

[5] Williams II et al. The virtual haptic back for palpatory training. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 191–197. ACM, 2004.

[6] M. Fritschi, M.O. Ernst, and M. Buss. Integration of Kinesthetic and Tactile Display–A Modular Design Concept. In *Proceedings of the EuroHaptics*, 2006.

[7] J.P. Fritz, T.P. Way, and K.E. Barner. Haptic representation of scientific data for visually impaired or blind persons. In *Proceedings of the Eleventh Annual Technology and Persons with Disabilities Conference*. CSUN, 1996.

[8] Novint Technologies Inc. *HDAL - Novint Falcon SDK*, 2008. `http://home.novint.com/products/sdk.php`.

[9] Kyle Machulis. *libNiFalcon - open source driver for the Novint Falcon*, 2009. `http://qdot.github.com/libnifalcon/index.html`.

[10] Kadleček Petr. A Practical Survey of Haptic APIs, Bachelor's thesis, Charles University in Prague, Czech Republic, 2010.

[11] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The haptic display of complex graphical environments. SIGGRAPH '97, pages 345–352, 1997.

[12] Kenneth Salisbury and Christopher Tarr. Haptic rendering of surfaces defined by implicit functions. In *Proceedings of the ASME 6th Annual Symposium*, pages 61–68, 1997.

[13] Sensable. *OpenHaptics software development toolkit* , 2011. `http://www.sensable.com/products-openhaptics-toolkit.htm`.

[14] C. B. Zilles. Haptic Rendering with the Toolhandle Haptic Interface. Master's thesis, Massachusetts Institute of Technology, 1995.

[15] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. IEEE Computer Society, 1995.

Real-time Hand Tracking using Flocks of Features

Bc. Andrej Fogelton*
Supervised by: Ing. Matej Makula, PhD.†

Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava / Slovakia

Abstract

There is a growing demand to interact with computers in a more natural way. For example using hand gestures to interact with certain type of applications would be more efficient than old-fashioned keyboard and mouse. Hand tracking is one of the key problems in computer vision. We have analyzed many different approaches used for hand tracking. Flocks of features introduced by Mathias Kölsch and Matthew Turk can track human hand continuously during various movements and pose variations. It uses the Kanade Lucas Tomasi (KLT) tracker for features located on a human hand to track them in a frame sequence. It can handle fast tracking of non-rigid highly articulated objects such as hands. We propose an improvement to this algorithm by processing the frame using histogram back projection of the skin color prior to applying flocks of features (FoF). This modification provides better results with lower false positive error.

Keywords: Hand Tracking, Flocks of Features, Back Projection, Histogram

1 Introduction

In the last few years, there is a growing demand to control computers in a more interactive way then using just mouse and keyboard. One of the pioneers of the new way of interaction was Nintendo WII,[1] which uses infrared LEDs and infrared camera with a proximity sensor. This device is used in a game console offering a totally new way of game experiencing. For example, you can play tennis by holding WII remote controller in your hand instead of your tennis racket and play a match against your friend or the computer.

The other promising product was introduced by Microsoft. The new version of their game console XBOX 360 uses *Kinect*,[2] which has the ambition to become even more popular than WII. Kinect is a webcam extended with infrared light camera and infrared light projector. This projector illuminates the scene with infrared light and a special infrared light camera is able to compute the depth information from the image. This information can be used to interact with computer like never before. You drive a car or play almost everything what you want and with this camera the computer is capable of creating a model of human figure in real-time. This opens new possibilities of interaction with the computer.

We believe that in several cases using hands to interact with the computer will be much more efficient than old-fashioned mouse or keyboard. We want to make this solution accessible for larger population by using a web-cam. In order to deal with highly articulated objects, such as hands, effectively in the most common situations with arbitrary background, several requirements were given:

- background invariant,
- without gloves or any other markers,
- light invariant,
- ability to track both hands of the user real-time,
- hand shape (pose) invariant,
- hand size invariant.

In Section 2, we describe the present state of art in the hand tracking area. The algorithm *Flocks of features* is described more precisely in Section 3. In Section 4, we present our modifications to this algorithm to overcome some of its difficulties. We discuss the results of different methods in Section 5. Finally, in the last section, we propose the conclusion and the future work.

2 Related Work

There are several methods to track human hands. A tracker can be based on a few cues or their combinations; shape (contour), color, and behavioral knowledge about the hand. A lot of algorithms work fine, however there are lots of restrictions to the user's behavior or background. The aim of good interface is not to restrict users but to allow them maximum freedom and comfort. There are summarizing papers [12, 10] which bring brief informations on several techniques used for hand detection and hand tracking.

*fogelton@gmail.com
†makula@fiit.stuba.sk
[1] http://www.nintendo.com/wii
[2] http://www.xbox.com/en-US/kinect

2.1 Color Based Tracking

One of the first ideas to detect human hand is to use the color based filter. Every human hand has a specific color. The basic principle is to use a set of threshold ranges for every image channel separately. This solution is limited to several conditions. There should be no other objects having the same skin color characteristics in the captured image, lighting conditions have to be constant and the person has to be of the specific skin color. For example black people do not have the same range of thresholds like white people. These restrictions make this solution not very useful, but there is a number of possibilities to improve it.

Color Models

The most widely used is the RGB color model, in which every color is described by the intensity of three basic colors: Red, Green and Blue. High correlation between these components and luminance mixing with the chromaticity makes this color model very sensitive to the light condition changes [3]. There are several other models, which have the intensity of colors in a separate channel. Most common are HSV, HSL, YCbCr or normalized RGB.

Simple Color Classification using Randomized Lists

An interesting solution was presented in the paper *Robust hand tracking using a simple color classification technique*, which uses even different color model called *L*a*b** [19]. It is also quite invariant to luminance conditions with separately defined luminance value *L** and chromatic values *a** and *b**. Threshold ranges are setup with sample pixels from a hand, but this is not the only classification method used. This solution is based on clustering similar color pixels and defining a region of interest during the initialization process (in our case a human hand). This solution presents very good tracking results (Figure 1) of hand under various luminance conditions and with almost no false positive cases. However, we believe that this solution will stop working when the hand comes in contact with face and the classifying method would fail.

Figure 1: Hand clusterization [19].

Mean-Shift and CamShift

Mean-Shift is a robust color segmentation method [1] based on selected region matching. It is converging from an initial guess for location and scaling to the best match based on the color histogram probability. CamShift detects the mode in the probability distribution by applying Mean-Shift and dynamically adjusting the parameters of the target distribution. Mostly it is working fine but with very bad size precision and objects with similar color can easily distract the tracker. It is the standard, the results can be compared to.

Modified CamShift [18] takes into consideration a kind of gray model, which also represent the forecast of change in the image sequence. Better results are achieved because of this modification and there is no distraction due to the contact with face (Figure 2).

Figure 2: No distraction with face using Gray CamShift tracking [18].

2.2 Background Subtraction

This technique is used mostly in [11] with fingertip detection. It can be used when the background is given or stable without any other motion (hand only). This restriction makes it applicable only to a few situations.

2.3 Contour Based

Another cue on which we can base tracking is the contour. There are several solutions that use specific shape of a hand posture for detection and tracking. The basic technique to enhance image is the *Canny* edge detector. A more sophisticated one is the Oriented edge energy (Figure 3), which is the result of several filters [15]. It is hard to define all possible contours of a human hand and that is why contours are mostly used for posture estimation on selected region.

Figure 3: Original image, Canny detector, Oriented edge energy [15].

Condensation

The condensation approach [5] models the probability distribution with a set of random particles and performs all involved calculations on this particle set. Condensation presents very good results with proposed hand posture in Figure 4, but it is not designed to track human hand while posture changes a lot.

Figure 4: Sample of condensation tracking [5].

2.4 Model Based Detection

This principle is always used with some other cues, where we can use knowledge about the hand. For example in [2] it is used with skin color probability map to detect the hand and its posture, which depends on how many fingers are straight. This kind of model can be useful when the hand is pointed to the camera and the number of straight fingers can be clearly seen.

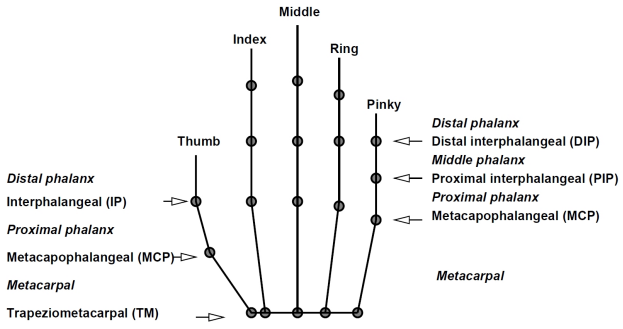


Figure 5: One of the skeleton hand models [12].

There are several types of hand models [12]. The skeleton model (Figure 5) is very common. They usually differ in number of points and vectors. The motion model [15] is another option, which can be used as another cue to extend the color based tracking.

3 Flocks of Features

Mathias Kölsch and Matthew Turk presented a *Fast 2D hand tracking with flocks of features and multi-cue integration* [7]. This algorithm can track the human hand without any artificial objects such as gloves. It is robust to various light conditions and furthermore a non stationary camera can be used. The tracker's core idea is motivated by the seemingly chaotic flight behavior of a flock of birds [13] such as pigeons. The minimum and maximum safe distance during the flight are defined. Features of the hand are also very close together like birds in a cloud [13]. The minimum distance between any two features and the maximum distance from the center (median) is defined. The median position of features is computed and the search using optical flow can be provide only up to the maximum distance from this position.

Listing 1: Flocks of features algorithm [8].

```
input:
bnd_box - rectangular area containing hand
mindist - minimum pixel distance between
    features
n - number of features to track
winsize - size of feature search windows

initialization:
learn color histogram
find n*k good-features-to-track with mindist
rank them based on color and fixed hand mask
pick the n highest-ranked features
//k=3 was used

tracking:
update KLT feature locations with image pyramids
     compute median feature
for each feature
  if less than mindist from any other feature
     or outside bnd_box, centered at median
     or low match correlation
  then relocate feature onto good color spot
      that meets the flocking
  conditions

output:
median - the average feature location
```

Robust hand detection [8] is used to initialize this method. Very good results are achieved during rapid movements and with continuous pose changing of the human hand. An overview of the entire algorithm is listed in Listing 1.

3.1 KLT Features

The KLT tracking algorithm calculates a brightness gradient (sobel operator) along at least two directions for a promising feature candidate to be tracked over time [14, 16]. In combination with image pyramids (a series of progressively smaller-resolution interpolations of the original image), a feature's image area can be matched efficiently to the most similar area within a search window in the following video frame. If the feature match correlation between two consecutive frames is below a threshold, the feature is considered "lost". A hand detection method supplies both a rectangular bounding box and a probability distribution to initialize tracking.

The probability mask states for every pixel in the bounding box the likelihood that it belongs to the hand. Features are selected within the bounding box according to their

ranking and observing a pair wise minimum distance. These features are being ranked according to the combined probability of their locations and color. Highly ranked features are tracked individually per frames. Their new locations become the area with the highest match correlation between the two frame's areas.

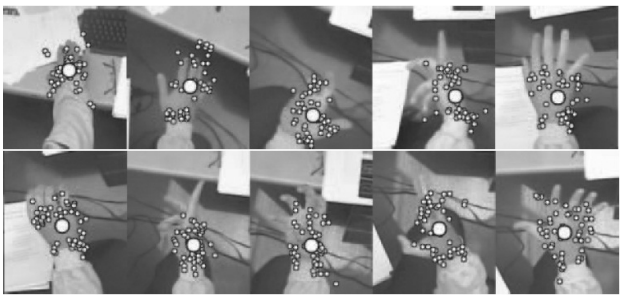


Figure 6: Snapshots of sequences with hand motions; the cloud of little dots are features and the big dot is their median [7].

Individual features can latch onto arbitrary artifacts of the object being tracked, such as fingers of a hand. Their movement is independent along with the artifact, without disturbing other features. Too dense concentrations of the features that would ignore other object's parts are avoided due to the minimum distance constraint. But stray features that are too far from the object of interest are brought back into the flock with the maximum distance constraint. To get more stable results, about 15% of the furthest features from median computation have to be removed. The speed of pyramid-based KLT (Kanade, Lucas, Tomasi) [14, 16] feature tracking allows to overcome the computational limitations of tracking the model-based approaches and achieving real-time performance.

3.2 Color Classification

During calibration process, a hand color is observed and the normalized-RGB histogram is created. Using this technique exclusively is not a very good solution because it can detect objects with similar color histogram such as wooden objects or other parts of the human body. The color information is used as a probability map. At tracker initialization time, the KLT features are placed preferably onto locations with high skin color probability. New location of a relocated feature is chosen with high color probability (more than 50%). Changing light condition can cause bad tracking performance, but only in case of relocated features because most of the features will continue to follow gray-level artifacts. This method combines cues from feature movement based on gray-level image texture with cues from texture-less skin color probability. It depends on the algorithm parameters how often features are relocated and on the importance of the color modality.

This algorithm was used to interact (Figure 7) with a wearable computer [9]. A webcam was placed at the head mounted display, so the hand size was approximately constant. It can be used to track both hands [4] or even other objects, where the skin color is replaced by a given sample. The problem is that it is not size invariant due to the threshold for the maximum distance from the center of the flock.

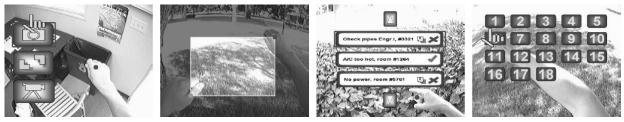


Figure 7: Screen shots from glasses of given application [9].

3.3 Flocks of Features with Appearance

New version of FoF was introduced, where another cue was added [6]. Haar features detector [17] was trained on a set of images of hands with different pose variations. This is used side by side with FoF. Positive detection of hand is evaluated after passing several stages (Figure 8). This adapted algorithm uses probability given not only by skin color, but also by the detector (the last passed stage of AdaBoost). The presented solution reaches better results than the original flocks of features in all tested cases [6].

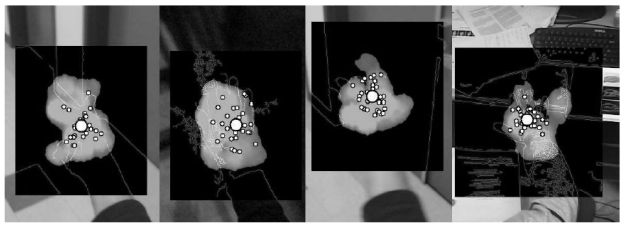


Figure 8: The appearance-based prior for selected hand images [6].

4 Modifications of Flocks of Features

The original FoF uses gray-level image for KLT features tracking. Due to this procedure we found the FoF algorithm to be vulnerable to edges occurring in the background. During movements over strong edges, a lot of KLT features can be relocated into incorrect positions. This leads to an incorrect median relocation and tracking failure (Figure 9).

We tried to avoid this kind of failure by ranking features based on skin color probability also during tracking and not only at the initialization procedure, but this did not lead to the expected results. We used histogram (calculated from a given region – hand palm) from the initialization procedure to create a probability map by applying

Figure 9: FoF algorithm is vulnerable to edges occurring in the background due to bad KLT features relocation.

back projection during tracking on every frame. One slight difference is use of the HSV color model instead of normalized-RGB, because it is more common in these kind of application while using histogram. The HSV color model and normalized-RGB have similar characteristics in terms of luminance invariance.

4.1 Image Processing

We realize that if we use the probability map instead of the original image, we will get rid of the edges in the background because they are not skin colored and they did not appear in the back projected image. The idea to run FoF on this probability map (Figure 10) has been proved to be a step forward. Look at the resulted image on Figure 10, there is a lot of noise. To reduce this noise we use basic image processing operations: erosion and dilation (Figure 11). From testing we can say that the best ratio *reducing noise/reducing skin regions* is achieved when applying erosion and then dilation, each one only once. This is also called the operation open.

Figure 10: Result of histogram back projection with noise.

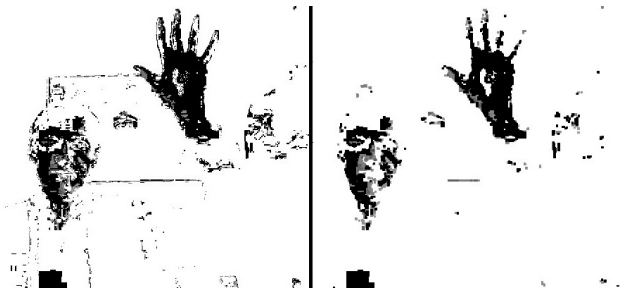


Figure 11: Removing noise with erosion and dilation.

Because of this modification we do not need to rank features in the initialization procedure and we can be almost sure that every feature will be located somewhere in the skin region. The modified algorithm is listed in Listing 2.

Listing 2: Modification of Flocks of features algorithm.

```
input:
bnd_box - rectangular area containing hand
mindist - minimum pixel distance between
     features
n - number of features to track
winsize - size of feature search windows

initialization:
learn color histogram
create back projected image
find n good-features-to-track with mindist

tracking:
create back projected image
update KLT feature locations with image pyramids
      compute median feature
for each feature
  if less than mindist from any other feature
     or outside bnd_box, centered at median
     or low match correlation
  then relocate feature onto good color spot
       that meets the flocking
  conditions

output:
median - the average feature location
```

5 Results

We present a sequence of images with tracking results (Figure 12). We considered tracking to be lost when the median came out of the hand palm for more than one second. Our modification can also fail like the original one (Figure 13), mostly because of rapid movements of the hand over face or other skin colored objects. An ordinary webcam is able to achieve 30 frames per second, but this is not enough for rapid movements. The reason for the KLT tracking failure in this case is the optimization to look for a new location of a given feature in range of 10 pixels (the bigger the range, the more time it takes to compute). That is the reason why it considers other skin colored parts as hand and the median is disrupted.

Another case of tracking failure is when the median slides off the hand palm (Figure 14), because the given person is not wearing a shirt with long sleeves. This tracking is based mostly on color cue and this disruption is very common using this principle. The solution is to add another cue which would represent the hand contour; it should be some general contour with help of which we could determine the border between hand palm and forearm. This could be our task in the future.

The HSV color model provides luminance invariation, but there are situations when the histogram back projection is not working well. For example, when a human hand is moving too far from the camera, the luminance from the

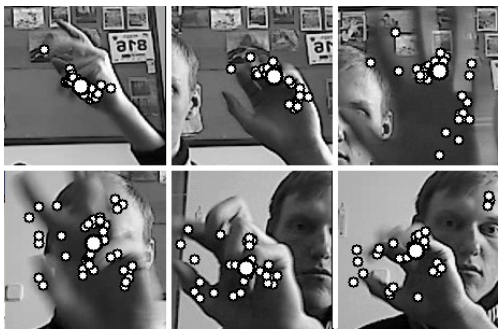


Figure 12: Example of hand tracking(200×200 image parts are cutted out from 640×480 images).

Figure 13: Tracking failure due to rapid movements.

hand changes rapidly and we can see only small parts of the hand (Figure 15) on the back projected image. This can also lead to tracking failure. This unpleasant effect can be also seen when the main source of light is not over head and luminance can change a lot by rotating the hand.

5.1 Comparison

In general it is hard to compare methods, as they have different advantages and disadvantages. We did some comparisons between CamShift, our implementation of FoF and our modification of FoF. We did not add new features when others where considered lost. Our aim is to objectively verify the ability of tracking various hand postures in a frame sequence while trying to disrupt the tracker by movements across face and the other hand.

We made 6 pairs of videos with different length from 400 to 1000 frames. Each pair consists of similar videos. One is with a man wearing long sleeves shirt ('a' labeled videos), the other wearing t-shirt ('b' labeled videos). Videos and tracking results can be downloaded from my website.[3] One pair was made outside, the others inside. They are aimed at different conditions like changing the size of the hand, rapid movements or moving background. Complete list of all tested videos can be found at Table 1. The results are processed in the graph (Figure 16), where the height of columns means the number of frames till tracking failure.

[3] http://henryi.yweb.sk

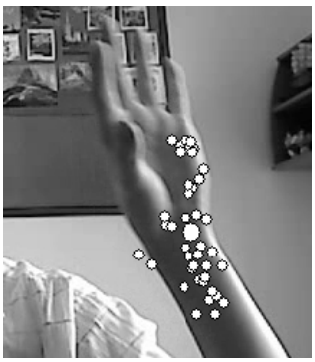


Figure 14: Median slided off the hand palm.

Figure 15: Example of bad back projection due to the luminance variation of the skin.

Conclusions and Future Work

We analyzed different hand tracking solutions with the aim to find a solution which could be used as an input to interact with computer. The main goal is not to restrict the user with behavior rules. It should be possible to track human hand without wearing any gloves or forcing the users to hold their hand in one posture all the time to get tired easily. Flocks of features showed up to be a very robust algorithm, but it can be easily disrupted with strong edges in the background. Our idea to process an image with histogram back projection led to a significant improvement in tracking efficiency and decreasing false positive KLT features tracking. But there are still several issues to handle.

We proofed, that running FoF on back projected image is better in almost all cases (Figure 16) than the original FoF and it can handle more reliable tracking than CamShift in all cases. In one case original FoF showed to be better due to bad luminance conditions (Back projection is not working well, when the luminance condition change a lot from initialization procedure.) and the gray-level image used for tracking is more suitable for these kind of

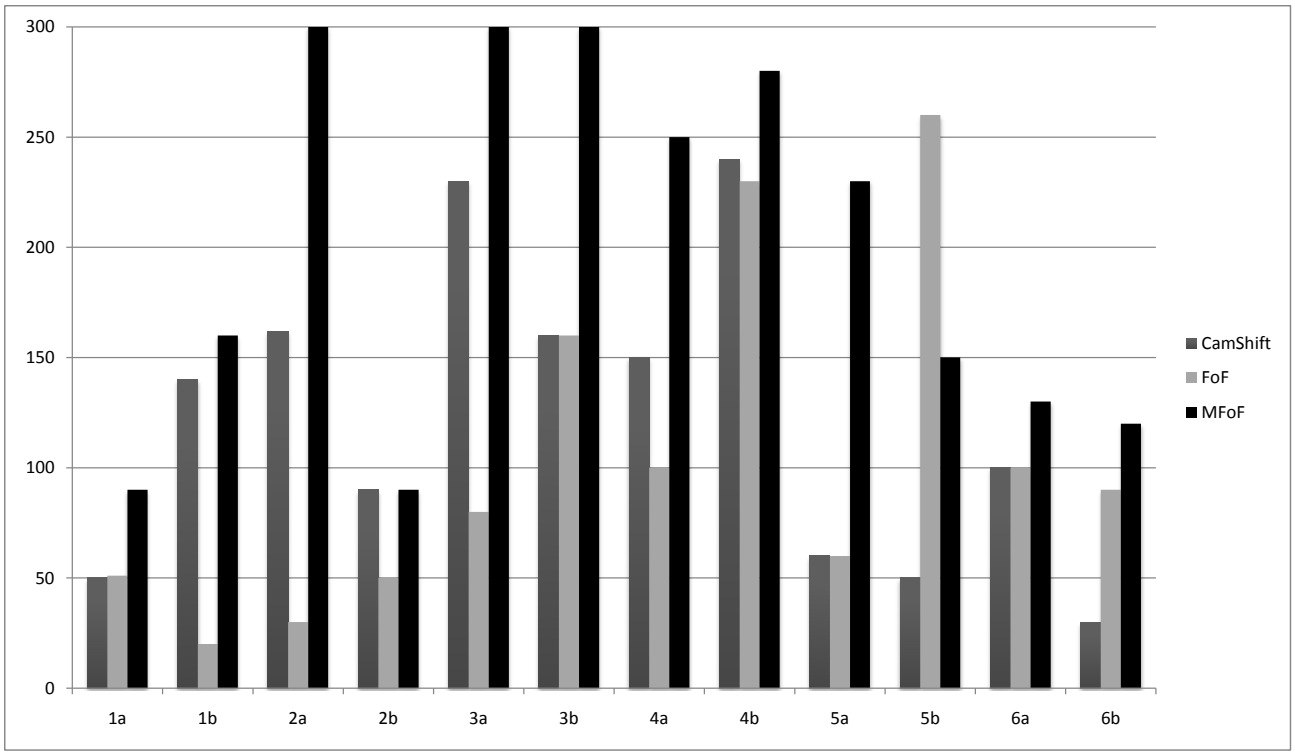


Figure 16: Testing results (Modified FoF was able to track the whole video sequence in cases: 2a, 3a, 3b).

number	conditions
1	rapid movements of the hand
2	size of the hand is changing a lot
3	arbitrary movements
4	arbitrary movements
5	moving background
6	outside lightining conditions

Table 1: Explanation of listed videos

situations.

Size Invariant Flocks of Features

Due to the maximum distance threshold constant, flocks of features has a problem with moving the hand closer and further from the camera. This can be avoided by finding proper algorithm which would calculate this threshold depending on the density of features.

Contour Cue

We want to add a third cue to the flocks of features which would represent the contour of the hand palm. This will help us to avoid the unpleasant situation when features are detected outside the palm and the median is distracted and moved outside the palm of the hand (our aim is to keep the median inside the hand palm).

Tracking Lost Detection

After adding the contour cue we can setup tracking failure detection, which will be based on the median coordinates. Tracking could be considered lost when the median is moved outside the hand palm contour.

Acknowledgements

This work was supported by grant KEGA 244-022STU-4/2010.

References

[1] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning opencv, 1st edition*. O'Reilly Media, Inc., 2008.

[2] Lars Bretzner, Ivan Laptev, and Tony Lindeberg. Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. In *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, FGR '02, Washington, DC, USA, 2002. IEEE Computer Society.

[3] Reza Hassanpour, Asadollah Shahbahrami, and Stephan Wong. Adaptive gaussian mixture model for skin color segmentation. *World Academy of Science, Engineering and Technology*, 41, 2008.

[4] Jesse Hoey. Tracking using flocks of features, with application to assisted handwashing. In *British Machine Vision Conference (BMVC)*, 2006.

[5] Michael Isard and Andrew Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[6] Mathias Kölsch. An appearance-based prior for hand tracking. In Jacques Blanc-Talon, Don Bone, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, volume 6475 of *Lecture Notes in Computer Science*, pages 292–303. Springer Berlin / Heidelberg, 2010.

[7] Mathias Kölsch and Matthew Turk. Fast 2d hand tracking with flocks of features and multi-cue integration. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*, pages 158 – 158, 27-02 2004.

[8] Mathias Kölsch and Matthew Turk. Robust hand detection. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 614 – 619, 2004.

[9] Mathias Kölsch, Matthew Turk, and T. Hollerer. Vision-based interfaces for mobility. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 86 – 94, august 2004.

[10] Fariborz Mahmoudi and Mehdi Parviz. Visual hand tracking algorithms. In *Proceedings of the conference on Geometric Modeling and Imaging: New Trends*, pages 228–232, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Shahzad Malik and Joe Laszlo. Visual touchpad: a two-handed gestural input device. In *Proceedings of the 6th international conference on Multimodal interfaces*, ICMI '04, pages 289–296, New York, NY, USA, 2004. ACM.

[12] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):677 –695, jul 1997.

[13] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, July 1987.

[14] Jianbo Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593 –600, 21-23 1994.

[15] Björn Stenger. Template-based hand pose recognition using multiple cues. In P. Narayanan, Shree Nayar, and Heung-Yeung Shum, editors, *Computer Vision ACCV 2006*, volume 3852 of *Lecture Notes in Computer Science*, pages 551–560. Springer Berlin / Heidelberg, 2006.

[16] Carlo Tomasi and T Kanade. Detection and tracking of point features. *Image Rochester NY*, pages Technical Report CMU–CS–91–132, April 1991.

[17] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511 – I–518 vol.1, 2001.

[18] Jiajun Wen and Yinwei Zhan. Vision-based two hand detection and tracking. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, pages 1253–1258, New York, NY, USA, 2009. ACM.

[19] Miaolong Yuan, Farzam Farbiz, Corey Mason Manders, and Ka Yin Tang. Robust hand tracking using a simple color classification technique. In *Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '08, pages 6:1–6:5, New York, NY, USA, 2008. ACM.

Natural Phenomena & GPU

Towards Supporting Volumetric Data in FurryBall GPU Renderer

Michal Benátský*
Supervised by: Jiří Bittner†

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

Abstract

This paper describes an implementation of volumetric rendering for FurryBall gpu renderer. Since FurryBall is directly integrated into Autodesk Maya, our volumetric renderer supports all types of fluids, which can be simulated in Maya. We discuss the issues of integrating volumetric rendering into the FurryBall renderer. We show that our implementation of volumetric rendering is up to 3 orders of magnitude faster than Mental Ray on the tested scenes.

Keywords: volumetric rendering, fluids, modeling tools

1 Introduction

Fast rendering preview integrated directly into 3D modeling tools, would significantly enhance the artists comfort and productivity. Most 3D modellers however do not provide high quality realtime feedback and rendering of scenes can take hours even days.

GPU based multi pass rasterization can be used to generate fast realistic previews and with some limitations it can also be used to generate final high quality results. One of such renderers is FurryBall [10], which is a GPU based rasterization renderer, directly integrated into Autodesk Maya.

In this paper we show how to extend the renderer for dealing with fluids. In particular we discuss how to integrate our volumetric renderer with the rasterization of transparent objects, shadows, hair systems, and how to deal with volumetric grid containers intersections. We present results obtained on several test scenes representing simulated fluids and show that these results are consistent with Mental Ray while obtaining a significant speedups.

2 Related Work

Rendering volumetric data is well studied area of computer graphics. For a comprehensive overview of realtime volumetric rendering techniques please refer to Hadwiger et al. [3].

*benatmic@fel.cvut.cz
†bittner@fel.cvut.cz

Figure 1: Church model render in FurryBall, model courtesy of Art and Animation studio

Pixar's RenderMan based on Reyes architecture [1] is known as industry's standard and is very widely used. Render Ants [13] shows that it is possible to move all stages of Reyes to the gpu. V-Ray is one of the leading renderers in the field of GPGPU raytracing and it is fully integrated into Autodesk Maya and 3ds Max. Another well known render is iray from Mental Images, creators of Mental Ray raytracer.

Apart from FurryBall we are aware of only a few commercially available renderers based on multi pass rasterization. The most similar is pixar's lpics [8] for realtime preview of lighting on 3D scene. Another related renderer is Mach Studio, which is a realtime GPU rasterization renderer. However these renderers aren't fully integrated into 3D modeling software such as Maya or 3Ds Max.

The rest of the paper is organised as follows: The basics of volumetric rendering and our implementation of rendering volumetric data will be described in the next section. In section 4 we will describe the integration of volume rendering into existing renderer based on multi pass rasterization on the gpu and the integration with Autodesk Maya fluid rendering as well. In section 5 we will present results of our work.

3 Volumetric Rendering

In this section we first describe the theoretical background of volume rendering and then outline our implementation.

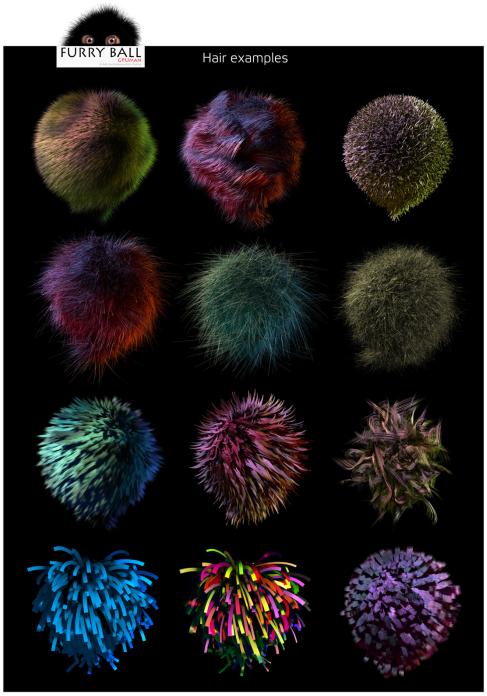


Figure 2: Fur rendering with FurryBall, model courtesy of Art and Animation studio

3.1 Theoretical background

The physical basis for volume rendering relies on geometric optics, in which is light assumed to travel along straight line unless interaction with participating media takes place [3]. The following types of interaction are typically taken into account.

Figure 3: Ray in volume with emission, absorption and scattering.

Emission Volume emits light and increasing the radiative energy. Practical example could be fire, which emits light by converting heat into light.

Absorption Volume absorbs light by converting radiative energy into heat.

Scattering Light can be scattered by volume, changing the direction of light propagation.

For solving complex light transport problem are commonly used simplified models. In "Absorption only" model volume absorb incident light. No light is emitted or scattered. "Emission only" model presents volume which is completely transparent, but can emit light. "Emission-Absorption" model is most common in volume rendering. Volume emits light and absorb incident light. "Single Scattering" model counts with single scattering from light that comes from external light source (not from the volume). "Multiple Scattering" model has goal to evaluate the complete illumination model for volumes.

Autodesk Maya uses "Emission-Absorption" model. Light scattering is approximated by shadow diffusion. Our renderer is implemented into Maya, so we use Emission-Absorption as well.

Emission-Absorption model can be described by Volume-Rendering Integral, which integrates radiance along the direction of light flow from the starting point $s = s_0$ to the endpoint $s = D$.

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{s} q(s) e^{-\int_{s_0}^{D} \kappa(t)dt} ds \qquad (1)$$

The term I_0 represents the light from the background. $I(D)$ is the radiance leaving the volume at $s = D$. κ is the *absorption coefficient* and q is *emission*. First term of the equation 1 describes the light from the background attenuated by volume. Second term represents the integral contribution of the source terms attenuated by the volume along the remaining distance to the camera.

$$\tau(s_1, s_2) = \int_{s_2}^{s_1} \kappa(s) ds \qquad (2)$$

τ defines *optical depth* between positions s_1 and s_2, which defines how long can light travel before it is absorbed. Smaller values defines material which is near to be transparent and higher values defines nearly opaque material. Transparency for a material between s_1 and s_2 is:

$$T(s_1, s_2) = e^{-\tau(s_1, s_2)} = e^{-\int_{s_1}^{s_2} \kappa(t)dt} \qquad (3)$$

Once we define transparency, we can rewrite volume rendering integral into:

$$I(D) = I_0 T(s_0, D) + \int_{s_0}^{s} q(s) T(s_0, D) ds \qquad (4)$$

Volume rendering integral cannot be solved analytically, that's why we use discretisation. Common approach is to split integration domain into n intervals: $s_0 < s_1 < ... < s_{n-1} < s_n$. Transparency and color contribution of the ith interval is:

$$T_i = T(s_{i-1}, s_i), c_i = \int_{s_{i-1}}^{s_i} q(s) T(s, s_i) ds \qquad (5)$$

The radiance at the exit point is:

$$I(D) = I(s_n) = I(s_{n-1})T_n + c_n = (I(s_{n-2})T_{n-1} + c_{n-1})T_n + c_n = \ldots$$

which can be rewritten as

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} T_j \tag{6}$$

with $c_0 = I(s_0)$. Which leads to recursive front to back equations

$$C_i = C_{i+1} + T_{i+1}C_i$$
$$T_i = T_{i+1}(1-\alpha_i)$$

and back to front equation:

$$C_i = C_{i-1}(1-\alpha_i) + C_i$$
$$T_i = T_{i-1}(1-\alpha_i)$$

α is opacity and $\alpha = (1-T)$.

Volume raycasting Volume raycasting [5] traces ray from the camera into the volume and solves volume rendering integral along these rays. The biggest advantage of volume raycasting is that rays are completely independent on each other and can be processed in parallel.

3.2 Implementation

We implemented two techniques for rendering the volumes. The first method is very simple and visualises the volume data as sprites (see Figure 4) and the second uses volume raycasting.

Particle rendering - splatting We represent every voxel of the volume as a vertex that is processed by a geometry shader to create a billboard. When the billboard is rasterized, the pixel shader samples the color and value ramps (transfer functions) to get appropriate color and opacity for every pixel. This part was not expected to produce high quality results, it was implemented only for the verification of received data from Maya and especially for the fast previews.

Volume raycasting Our second method uses volume raycasting written in HLSL on the pixel shader. It was first design decision to use direct compute, which is GPGPU part of the DirectX 11 API, but the nature of volume raycasting is allowing to implement it on the pixel shader, because of no need of synchronisations.

The rays are created based on the camera parameters. We support variable focal points and off-axis stereo camera. Ray origin and direction are converted into texture space using inverted world matrix. Converting to texture space allows faster raymarching and also it allows to perform a fast AABB test, because fluid container is a unit cube in texture space. This tells us minimum and maximum distance where to sample on the ray.

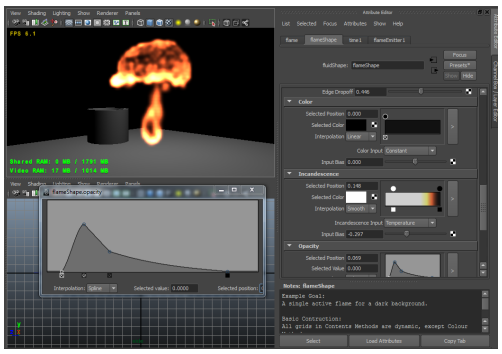


Figure 4: Fluid rendered using splatting in FurryBall

The sampling has a fixed step, but depth jitter can be used. Sampling is performed in parallel in the world space and in the texture space. The world space position is important for connecting to existing and optimized FurryBall shaders, especially for getting attenuation, light color (which can be defined as a texture) and shadow.

Note that our current implementation of volumetric ray casting fully replaced the splatting based approach as it can render a more accurate preview even faster (when using under sampling).

4 Integration with FurryBall and Autodesk Maya

FurryBall is a GPU based rasterization plug-in for Autodesk Maya [10]. FurryBall's initial purpose was a fast preview for artists, before raytracing a CG movie. As its development continued, the rasterization possibilities were able to suit most of the artist's needs and therefore it was extended from a simple previewer into a complete production renderer. FurryBall is written in C++, using DirectX 11 API and Open Maya API. It uses also Python and Maya Embedded Language (MEL). See Figure 1 and 2 for examples of images rendered with FurryBall.

4.1 Integration with FurryBall

Camera settings FurryBall supports all Autodesk Maya's camera settings, which means that implemented volume raycaster must support all this settings too and build rays correctly. We solved this problem by reconstructing near and far plane in world space using viewport, view and projection matrix and passing them to the shader. Then we interpolate between corner values and find correct ray origin and direction.

Opaque objects FurryBall allows to render all polygonal meshes and apply various textures such as color textures, bump textures and displacements textures. It also supports Maya layered textures and realtime adaptive DirectX 11 HW tessellation with displacement textures.

Volume raycasting can be combined with opaque geometry easily by stopping sampling at z-buffer value. Problematic part if this approach is when the size of the final render is not equal with the size of fluid render pass. This is supported because gaseous phenomena basically doesn't have hard edges and can be rendered in lower resolution. It is possible to have fluid render pass at 1/2 resolution of final render without visible quality loss. However this produces artifacts if geometry is inside volume, if no lower resolutions are possible to use even for final rendering.

Transparent objects Transparent objects in FurryBall are rendered with depth peeling [2] , with the possibility to set up the number of layers.

Integration of volumetric objects with transparent geometry is problematic, especially when geometry intersects fluid container, there is need to blend objects into the fluid in correct sample.

Two approaches were considered. First use customised depth peeling which take into account fluid rendering. That would solve the problem of sorting and also would solve problem of transparent object and fluid intersection by splitting fluid sampling into more parts.

Another considered approach was using order independent transparency presented by AMD at GDC 2010 [7], which uses unordered access writes into the output memory and outputs linked list for every rendered fragments containing informations about depth and color. This linked list is then sorted per fragment on the compute shader, blended with each other in correct order and then with the final render. This should be combined with fluid rendering by passing ordered lists to the fluid rendering pass.

Shadows for meshes FurryBall integrates various shadow mapping techniques. Basic shadow mapping techniques, including basic shadow mapping for spot and point lights and cascade shadow maps for directional lights [6]. [11]. For computing simplified soft shadows it uses PCF or more advanced PCSS [6], which can either use regular, Perlin or Poisson disk sampling.

It is not hard to combine our volumetric rendering with shadow mapping. Ray sampling in raycaster knows world position of the sample, so it can ask shadow map for shadows.

Reflections and planar mirrors General reflections and refractions and computed using environment mapping. Planar mirror reflections use rendering into a texture from the reflected camera position.

Both of these methods are not problematic to combine with described volume raycasting as rays can be created from the viewport and camera matrices.

Hair systems FurryBall has its own hair rendering system. It is based on constructing billboards or regular geometry along curves on geometry shader. This might be slower than a solution with lines, but it offers much more control and possibilities. Such as vegetation rendering using textured fur. Curves can be fully independent of Maya or can be connected to Maya hair system and benefit from Maya hair simulation.

Fluid integration with the hair rendering is the same as with regular geometry and rays stops at z-buffer value.

Shadow for hair and fur Shadows for hair can be computed by common shadow mapping techniques, which, however, do not provide sufficient quality. Tiny objects like hair appear much better when being shadowed with a transition function. Therefore FurryBall implements Deep Opacity Maps [12] and Fourier Opacity maps [4] for these cases.

Self shadowing and casting shadows for fluids is using Fourier Opacity maps, which allows us to save multiple hair systems and fluid containers into one map and cast shadows one to another and on the geometry.

Fluid containers intersections Fluid container intersections is common case in Autodesk Maya rendering. It is used for create sky, where one container simulates blue atmosphere and second contains clouds. Problem of fluid intersections is that samples has to be blended together correctly.

More approaches was considered. First one was very similar to depth peeling, where volume sampling is done in layers and layers are blended together. This wasn't accepted as too slow. We decided that FurryBall will sample two fluids together in one pass, if they intersects each other. This solution is faster and more accurate than making slices.

4.2 Integration with autodesk Maya

Maya offers very complex fluid solver, which is based on Navier-Strokes equations. The solver can handle both 2D and 3D grids and simulates density, temperature, speed, fuel and pressure. It also supports user defined gradients and constants in the simulation.

For fluid shading the most important are the *ramps* for colors and other values. Ramps define transfer functions by piece-wise linear functions. The user can connect one of the functions (like density or temperature) with one of the shading ramps (color, incandescence and opacity). It is also possible to connect gradients. Every ramp also has a specified bias. Color ramp defines the RGB transfer function which is affected by each light. Incandescence is also color transfer function which defines emitted light from

the volume. And at last is there opacity ramp which defines opacity transfer function.

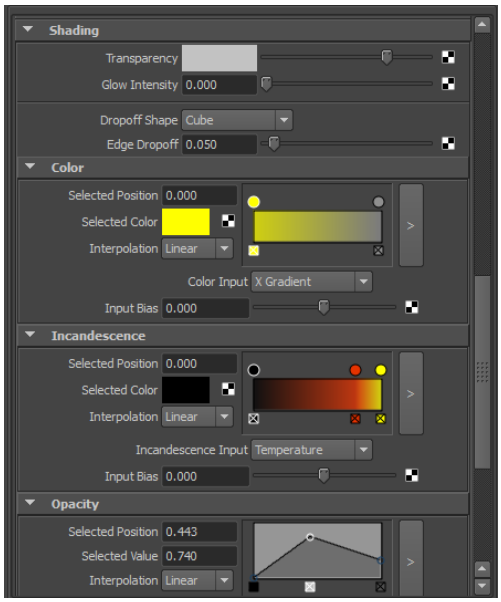


Figure 5: Fluid shading setup in Autodesk Maya

The final appearance depends on several additional settings. The user can define RGB transparency, which affects final opacity and color and also can choose some of the dropoff functions. User also define the number of samples per voxel and choose whether the fluid in container will receive/cast shadows. Maya light linking is also available (linking lights with scene objects to determine which one is lighted with which light). User chooses volume resolution and size, which affects final density, regular scale doesn't.

Fluid can be affected by procedural noises. Perlin [9], Billow, Volume wave and Wispy noises are employed. Lots of setting is possible. Coordinates can be fixed with volume grid or can flow fluid simulation by using velocity vectors. In FurryBall fluid renderer is currently Perlin and Billow noise implemented.

Fluids can receive and cast shadows. Self shadowing of fluid can be turned off for better performance of rendering. For better performance is also possible set fluid to do not interact with all lights in scene (or linked lights) but only one defined via fluid node gui.

Fluid node also enables to choose number of sample per voxel and interpolation method (linear or Smooth - cubic).

Our fluid renderer reads all ramps through the Maya API, samples them into a 2D texture. The attached functions such as density or temperature are loaded into 3D textures. The gradients are not loaded as they are computed directly in a shader.

	FurryBall	Mental Ray	speed up
no shadows	11ms	12000ms	1009 ×
shadows	36ms	18000ms	534 ×
shadows and noise	96ms	26000ms	279 ×
fire	96ms	6000ms	62 ×

Table 1: Rendering performance of FurryBall compared to Mental Ray. For rendered images see Figure 6. Image resolution was 800x600px. Scene with percolator contained 12000 triangles and fluid container with resolution $10\times10\times10$. And scene with fire contained 300 triangles and fluid container with $35\times35\times35$ resolution.

5 Results

FurryBall is able to render all of the maya fluids and provide same or very similar results as built-in renderers. In this section we will present several images rendered with FurryBall with their render times. Table 1 shows the rendering performance of FurryBall compared to Mental Ray, which we used as reference, on 4 different scenes. Figure 6 shows scenes referred in table 1. Figure 7 shows clouds created using perlin noise and Figure 8 shows scene with homogenous volume and two spot lights.

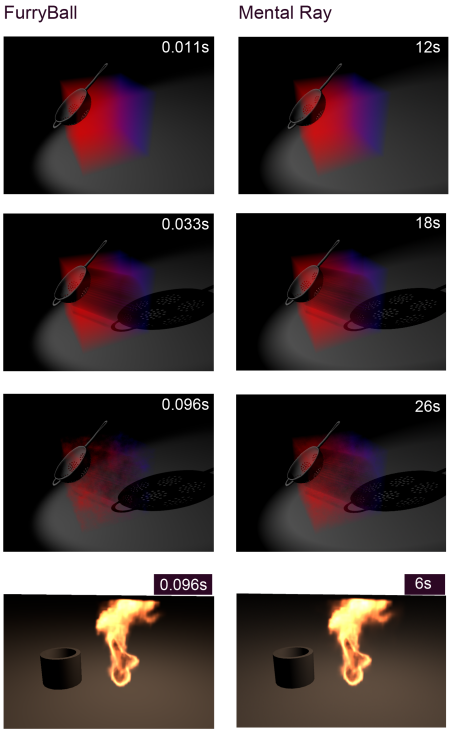


Figure 6: Images with render times. Left column: rendered with FurryBall. Right column: rendered with Mental Ray

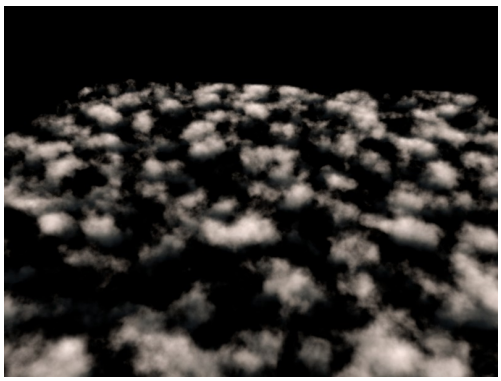


Figure 7: Procedural clouds rendered in FurryBall

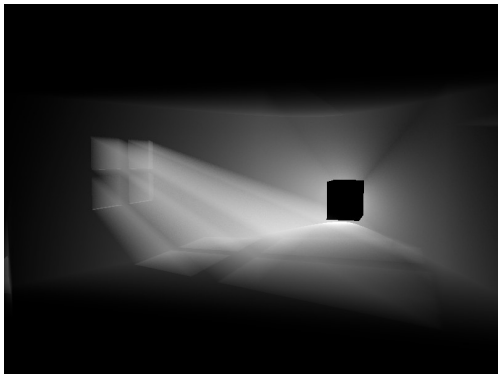


Figure 8: Volume with constant density, two lights are in scene

6 Conclusion

FurryBall became an interesting alternative to traditional renderers. It can be used for previsualization and realtime feedback before raytracing, but also as a final renderer, which can still provide results 50 - 300 times faster than traditional renderers. It is current being used for production of a new feature movie. Currently the fluid rendering plugin for FurryBall supports ray marching with emission and absorption and without restricting the user it produces the same or very similar results as reference built-in renderers.

References

[1] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. *SIGGRAPH Comput. Graph.*, 21:95–102, August 1987.

[2] Cass Everitt. Interactive order-independent transparency, 2001.

[3] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[4] Jon Jansen and Louis Bavoil. Fourier opacity mapping. In Daniel G. Aliaga, Manuel M. Oliveira, Amitabh Varshney, and Chris Wyman, editors, *SI3D*, pages 165–172. ACM, 2010.

[5] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8:29–37, May 1988.

[6] Mahdi Mohammadbagher, Jan Kautz, Nicolas Holzschuch, and Cyril Soler. Screen-space percentage-closer soft shadows. 2010.

[7] Holger Grn Nick Thibieroz. Oit and gi using dx11 linked lists. AMD, 2010.

[8] Fabio Pellacini, Kiril Vidimce, Aaron E. Lefohn, Alex Mohr, Mark Leone, and John Warren. Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Trans. Graph*, 24(3):464–470, 2005.

[9] Ken Perlin. Improving noise. *ACM Trans. Graph.*, 21:681–682, July 2002.

[10] FurryBall renderer. http://furryball.aaa-studio.eu. Art and Animation studio, 2010.

[11] Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. Sample-based visibility for soft shadows using alias-free shadow maps. December 10 2008.

[12] Cem Yuksel and John Keyser. Deep opacity maps. *Comput. Graph. Forum*, 27(2):675–680, 2008.

[13] Kun Zhou, Qiming Hou, Zhong Ren, Minmin Gong, Xin Sun, and Baining Guo. Renderants: interactive reyes rendering on gpus. Number 5, pages 155:1–11, New York, NY, USA, 2009. ACM. SIGGRAPH Asia 2009.

Sparse-Matrix-CG-Solver in CUDA

Dominik Michels*
Supervised by: Stefan Hartmann†

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

Abstract

This paper describes the implementation of a parallelized conjugate gradient solver for linear equation systems using CUDA-C. Given a real, symmetric and positive definite coefficient matrix and a right-hand side, the parallized cg-solver is able to find a solution for that system by exploiting the massive compute power of todays GPUs. Comparing sequential CPU implementations and that algorithm we achieve a speed up from 4 to 7 depending on the dimension of the coefficient matrix. Additionally the concept of preconditioners to decrease the time to find a solution is evaluated using the SSOR method. In the end additional suggestions are provided to further increase the speed of the presented CUDA cg-solver.

Keywords: parallized GPU solver, sparse matrix solver, conjugate gradient, ELLPACK-R, NVIDIA CUDA, SSOR precondition, 2D heat equation

1 Introduction

In several applications one has to solve linear equation systems with real, symmetric and positive definite coefficient matrices. Examples for such systems are broadly available e.g. physical deformation (cf. [6]), implicit mesh smoothing, mesh parameterization (cf. [9]), diffusion equation for terrain generation (cf. [8]). Usually linear equation systems are derived from dicretizing a continous problem resulting in very sparse coefficient matrices because only a small number of neighboring elements take influence on a specfic element. Therefore coffecient matrices can be stored very efficiently using sparse matrix formats e.g. the ELLPACK-R format which is used here. Such linear systems can become very large regarding to the given problem and one would like to find a time efficient solution for such systems. In literature the conjugate gradient algorithm is suggested to solve such linear, symmetric and positive definite systems. Specific operations of that algorithm can be parallelized e.g. scaled vector addition, dot product and matrix-vector multiplication. In this work a parallel implementation of the conjugate gradient algorithm using the NVIDIA CUDA architecture is presented to exploit the massive compute power of todays GPUs. Additionally the performance of the algorithm is compared to sequential and parallel implementations. Finally the 2D heat equation is solved using the parallized cg-solver.

*michels@uni-bonn.de
†hartmans@cs.uni-bonn.de

2 Related Work

The conjugate gradient algorithm was introduced in [7] as an efficient method to solve linear equation systems with real, symmetric and positive definite coefficient matrices. Additional details regarding the conjugate gradient algorithm can be found in [1], [11] and [12]. With the appearance of programmable graphics hardware a cheap way for getting massive parallel processors to the masses became possible. Therefore, general people can also take advantages from parallel algorithms. When analysing the conjugate gradient algorithm one will recognize that serveral operations within one iteration of the algorithm can be computed in parallel. In [5] a parallel implementation was introduced using GLSL shader programs. The necessary data for the computation was stored in textures and the algorithm was implemented in a pixel shader. Due to the general purpose GPU paradigm encoding of data in textures got superfluous because new technologies, like the NVIDIA CUDA architecture allow programming the parallel processors using an extended version of the C programming language (see [10]). An exemplary implementation of the conjugate gradient algorithm using CUDA is shown in [2]. If one wants to reach a time efficient implementation of the conjugate gradient algorithm it is essential to have an efficient way to compute sparse matrix vector products. In [4] an efficient storage for sparse matrices the ELLPACK format was suggested, which was extended to the ELLPACK-R format in [13] (explained later) especially for the use on GPUs. For time efficient solution of linear systems it is not enough to have efficient data structures and a lot of compute power: There exist extensions to the conjugate gradient algorithm which use a preconditioning matrix which is applied to the system matrix to decrease its condition number. In this paper a sequential version of such preconditioner is evaluated using the SSOR method described in [3].

3 Conjugate Gradient Algorithm

The primary introduction of the conjugate gradient algorithm from 1952 can be found in [7]. This method is used to solve linear systems $Ax = b$ with $A \in \mathbb{R}^{n\times n}$ (symmetric, positive definite) and $b \in \mathbb{R}^n$.
In this case the solution of the linear system is equivalent to the minimum of the function

$$E : \mathbb{R}^n \to \mathbb{R}, x \mapsto \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle,$$

meaning x solves $Ax = b$ if and only if E has a global minimum at x.
To proof the equivalence calculate $\nabla E(x) = Ax - b$ and $\nabla^2 E(x) = A$. Therefore, $\nabla E(x_0) \stackrel{!}{=} 0 \Leftrightarrow Ax_0 = b$ and $\nabla^2 E(x_0)$ is positive definite, wherefore x_0 is a local minimum. This is the only extremum, so x_0 is also a global minimum. This equivalence is the basic idea of the conjugate gradient algorithm. Instead of solving a linear system in a typical way, we search the minimum of the function E. Let $x = x^{(0)} \in \mathbb{R}^n$ be an arbitrary start vector. We search the miminum of E on the line

$$g : \mathbb{R} \to \mathbb{R}^n, \alpha \mapsto x + \alpha p.$$

The search direction p is arbitrary for now. Let $r := b - Ax$ be the residual. With $A = A^t$ we get

$$\frac{dE(g(\alpha))}{d\alpha} = -\langle r, p\rangle + \alpha\langle Ap, p\rangle$$

and with $dE(g(\alpha))/d\alpha \stackrel{!}{=} 0$

$$\alpha = \frac{\langle r, p\rangle}{\langle Ap, p\rangle}. \tag{1}$$

That is a minimum, because A is positive definite and we get

$$\frac{d^2E(g(\alpha))}{d\alpha^2} = p^tAp > 0.$$

To obtain the minimum approximately we use an iterative search with different search directions. For that purpose set an arbitrary start vector x and calculate a more precise approximation of the minimum in every iteration

$$x^{(m)} \leftarrow x^{(m-1)} + \alpha^{(m)} p^{(m-1)}.$$

To calculate α we need r and p. The residuals $r^{(m)} = b - Ax^{(m)}$ can be computed iteratively using

$$r^{(m)} \leftarrow r^{(m-1)} - \alpha^{(m)} Ap^{(m-1)},$$

because

$$\begin{aligned} & r^{(m-1)} - \alpha^{(m)} Ap^{(m-1)} \\ = \; & b - A\left(x^{(m-1)} + \alpha^{(m)} p^{(m-1)}\right) \\ = \; & b - Ax^{(m)} \\ = \; & r^{(m)}. \end{aligned}$$

We get the gradient descent algorithm (see Algorithm 1), if we choose the direction of the steepest descent $p = -\nabla E(x)$ as our search direction (cf. [1]). We set $x^{(0)} = 0$ and get $r^{(0)} = b - Ax^{(0)} = b$.

$x^{(0)} \leftarrow 0$
$r^{(0)} \leftarrow b$
$p^{(0)} \leftarrow r^{(0)}$

for $m \leftarrow 1$ **to** m_{max} **do**
  $\alpha^{(m)} \leftarrow \left\langle r^{(m-1)}, p^{(m-1)}\right\rangle / \left\langle Ap^{(m-1)}, p^{(m-1)}\right\rangle$
  $x^{(m)} \leftarrow x^{(m-1)} + \alpha^{(m)} p^{(m-1)}$
  $r^{(m)} \leftarrow r^{(m-1)} - \alpha^{(m)} Ap^{(m-1)}$
  $p^{(m)} \leftarrow -\nabla E(x)$
return $x^{(m_{max})}$

Algorithm 1: Gradient descent.

The convergence of this algorithm is a problem, because this method uses search directions, which are similar to each other. Resulting we get an increased number of iterations to reach sufficient accuracy. So we use a set of linearly independent search directions (A-conjugated directions). This seems to be a good approach, because the algorithm is able to minimize in every direction of the space $\mathbb{R}^n$ in n steps.
Two vectors $x_i, x_j \in \mathbb{R}^n$ are called A-conjugated ($x_i \perp_A x_j$) to a symmetric, positive definite matrix $A \in \mathbb{R}^{n\times n}$,
if $\left\langle x_i, Ax_j\right\rangle = 0$ (cf. [1]). A set $\{r_1, r_2, ..., r_k\}$ with $r_1, r_2, ..., r_k \in \mathbb{R}^n$ and $x_i \perp_A x_j$ for all $i \neq j$ is linearly independent.
The proof is easy. For all $l \in \{1, 2, ..., n\}$ with $\forall i \neq l$: $r_l^t A r_i = 0$ is

$$\begin{aligned} & 0 \stackrel{!}{=} \sum_{i=1}^{k} \alpha_i r_i \\ \Rightarrow \quad & 0 \stackrel{!}{=} r_l^t A \sum_{i=1}^{k} \alpha_i r_i = r_l^t A \alpha_l r_l = \alpha_l (r_l^t A r_l) = 0 \\ \Rightarrow \quad & \alpha_l = 0, \end{aligned}$$

because A is positive definite and therefore $r_l^t A r_l \neq 0$.
The search directions can be created iteratively. With the approach

$$p^{(m+1)} = r^{(m+1)} + \beta^{(m+1)} p^{(m)} \qquad \text{and} \qquad p^{(0)} = r^{(0)}$$

and the request $p^{(m+1)} \perp_A p^{(m)}$ we get

$$\left\langle r^{(m+1)}, Ap^{(m)}\right\rangle + \beta^{(m+1)} \left\langle p^{(m)}, Ap^{(m)}\right\rangle \stackrel{!}{=} 0$$

and therefore

$$\beta^{(m+1)} = -\frac{\left\langle r^{(m+1)}, Ap^{(m)}\right\rangle}{\left\langle p^{(m)}, Ap^{(m)}\right\rangle}. \tag{2}$$

With this selection of $\beta^{(m+1)}$ we get a new search direction $p^{(m+1)}$, which is A-conjugated to the old direction $p^{(m)}$. Terms (1) and (2) can be written in the advantageous form

$$\alpha^{(m+1)} = \frac{\left\langle r^{(m)}, r^{(m)} \right\rangle}{\left\langle p^{(m)}, Ap^{(m)} \right\rangle} \text{ and } \beta^{(m+1)} = \frac{\left\langle r^{(m+1)}, r^{(m+1)} \right\rangle}{\left\langle r^{(m)}, r^{(m)} \right\rangle}$$

(cf. [11]). So we save the computation of a dot product. Using A-conjugated search directions we get the conjugate gradient method (see Algorithm 2 cf. [12]). The output of the algorithm is shown for $n = 2$ in Figure 1.

$x^{(0)} \leftarrow 0$
$r^{(0)} \leftarrow b$
$p^{(0)} \leftarrow r^{(0)}$

for $m \leftarrow 1$ **to** n **do**
  $\alpha^{(m)} \leftarrow \left\langle r^{(m-1)}, r^{(m-1)} \right\rangle / \left\langle p^{(m-1)}, Ap^{(m-1)} \right\rangle$
  $x^{(m)} \leftarrow x^{(m-1)} + \alpha^{(m)} p^{(m-1)}$
  $r^{(m)} \leftarrow r^{(m-1)} - \alpha^{(m)} Ap^{(m-1)}$
  $\beta^{(m)} \leftarrow \left\langle r^{(m)}, r^{(m)} \right\rangle / \left\langle r^{(m-1)}, r^{(m-1)} \right\rangle$
  $p^{(m)} \leftarrow r^{(m)} + \beta^{(m)} p^{(m-1)}$
return $x^{(n)}$

Algorithm 2: Conjugate gradient (cg).

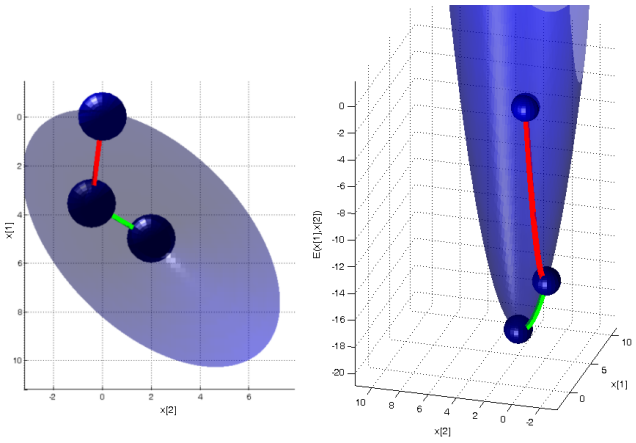


Figure 1: Cg-demo with $n = 2$.

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, b = \begin{pmatrix} 8 \\ -1 \end{pmatrix}, x^{(0)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, x^{(1)} = \begin{pmatrix} 3.5616 \\ -0.4452 \end{pmatrix}, x^{(2)} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$$

Using induction shows, that these n search directions are pairwise A-conjugated and therefore pairwise linearly independent.
By using this result it is possible to show that with exact arithmetic the method finds the solution x of the linear system $Ax = b$ after at most n steps (cf. [1] and [12]).
The worst case runtime is $\mathcal{O}\left(n^3\right)$ FLOPS, because we need the matrix-vector multiplication $Ap^{(m-1)}$ with runtime $\mathcal{O}\left(n^2\right)$ in every iteration. The other operations can be realized in linear time. It is recommended to calculate only one matrix-vector multiplication per iteration and store the result.

For sparse matrices exist adapted data structures to accelerate the matrix-vector multiplication. With parallelization it is possible to decrease the runtime.

In some cases we need less than n iterations to get a precise approximation of the solution. We use the 2-norm of the residual after every iteration to decide if more iterations are necessary. The 2-norm can be derived from $\langle r, r \rangle$ which is already calculated to determine β.
There again it makes sense to run more than n iterations to minimize the numerical error in some cases.

4 Compute Unified Device Architecture (CUDA)

The parallelization of the algorithm is done using the NVIDIA CUDA technology. This technology makes it possible to exploit the massive compute power of todays GPUs for general purpose computing by creating kernel programs which execute in parallel.

4.1 CUDA-Programing Paradigm

Next the CUDA programming paradigm is reviewed, for additional details the reader is referred to [4] and [10]. In this paper the CUDA Toolkit 3.1 is used. As mentioned above using the CUDA technology we gain the possibility to exploit the massive compute power of modern GPUs by writing kernel programs e.g. in an extended version of C and execute these programs N times in parallel on the available hardware. The program is executed by N different threads while no assumption can be made in which order the threads are executed. Started from the CPU a kernel is attached to a compute grid, which is separated into a number of blocks. In each block a specific amount of threads is running (see Figure 2). The threads of a

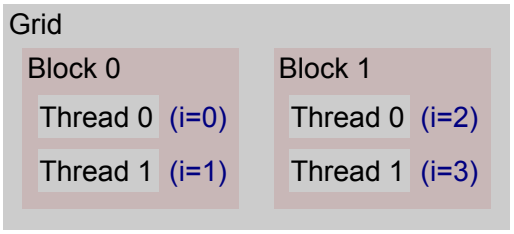


Figure 2: CUDA-programming model.

block are executed as packages of 16 threads which is also called halfwarp. The threads inside a halfwarp are generally running in parallel. The blocks of a compute grid are executed sequentially but in case of available hardware resources blocks are dispatched and executed in parallel to

other blocks. Every thread has two information: first in which block it is running and secound by which block internal number it is identified. Hence the numbering of the threads can be done by

```
int i = blockIdx.x * blockDim.x + threadId.x.
```

This is also illustrated in Figure 2. The next example demonstrates the addition of two vectors implemented in C for CUDA (see Algorithm 3).

```
01 __global__ void VecAdd(float* A, float* B, float* C, int n){
02     int i = blockIdx.x * blockDim.x + threadId.x;
03     if (i < n) C[i] = A[i] + B[i];
04 }
```

Algorithm 3: CUDA-vector addition
$C = A + B$ mit $A, B, C \in \text{float}^{256}$.

The data transfer between the CPU and GPU is done by *cudaMemcpy* which allows the transfer of data either from the CPU to the GPU or vice versa (*cudaMemcpyHostToDevice*, *cudaMemcpyDeviceToHost*). Within a kernel program one can only operate on data available in the GPUs memory. Normally the data is first stored in the global persistent GPU memory. This memory can be allocated by *cudaMalloc* and deallocated by *cudaFree* similar to the C programming language. The global thread identifier is stored in the local thread memory which is only available during the life time of a thread. Additionally the hardware provides shared memory which can be accessed very fast compared to the data access in the global memory. A specific compute kernel is initiated by the CPU with the information how many blocks inside the compute grid should be allocated and how many threads per block shall be spawned (see an example for the provided vector addition kernel below).

```
VecAdd <<< blocksPerGrid, threadsPerBlock >>>
                (A, B, C, 256);
```

A synchronization of the GPU with the CPU is possible by using *cudaThreadSynchronize*.

5 Parallelized Conjugate Gradients

In every iteration of the cg-algorithm we have to compute several scaled vector additions, dot products and a matrix-vector multiplication (see Algorithm 2).

5.1 Parallel scaled Vector Addition

The λ-scaled vector addition

$$sum = x + \lambda y$$

of two vectors each with n components can be realized with n threads, which execute in parallel. Every thread calculates one component of the result. This method equates to Algorithm 3 with additional λ-scaling. If t threads run in parallel the runtime can be decreased from $\mathcal{O}(n)$ to $\mathcal{O}(n/t)$.

5.2 Parallel Dot Product

The calculation of the dot product

$$dot = \langle x, y \rangle$$

of two vectors each with n components is realized in two steps. First we have to calculate a vector

$$help = x.*y,$$

which contains the pointwise product of the vectors. This procedure is similar to Algorithm 3, but we have to replace the addition by a multiplication. The acceleration is analog a decrement from $\mathcal{O}(n)$ to $\mathcal{O}(n/t)$.
In the second step we have to sum up all components of the vector $help$. The complexity of the sequential method is in $\mathcal{O}(n)$. To parallelize the calculation we have to sum up all neighboring components first ($\approx n/2$ FLOPS) and repeat this kind of summation with the resulting sums. This procedure can be realized iteratively ($\approx \log_2 n + 1$ iterations) and is pictured in Figure 3. Let t be the number of

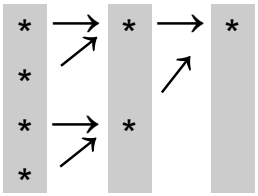


Figure 3: Parallel summation.

threads which run in parallel. Every thread calculates one sum of neighboring components. So the complexity for the second step and the whole dot product calculation is in $\mathcal{O}(n \log n/t)$.
If the number of threads is significantly lower it is more efficient to sum up not only two neighboring components. One option is that every thread has to add n/t elements first. So we get a new vector with t components, which has to be summed up. This can be done in $\mathcal{O}(\log t)$ and we get a complexity in $\mathcal{O}(n/t + \log t)$ for the second step and the dot product calculation. In this paper the first method is used.

5.3 Parallel Matrix-Vector Multiplication

The product of a $n \times m$-matrix and a vector with m components can be realized with n threads, where every thread calculates a component of the result.
This strategy is not efficient, if we want to calculate the product with a sparse matrix, because we execute many unessential operations with zeros. An approach to accelerate the calculation is using a special data structure to store the matrix.

5.3.1 ELLPACK-R-Data Structure

The classical formats to store a sparse matrix (coordinate storage, compressed column storage, compressed row

storage) are not applicable to parallelize the matrix vector product (e.g. [13]). In [4] an efficient storage for sparse matrices the ELLPACK format was suggested, which was extended to the ELLPACK-R format in [13] especially for the use on GPUs. A matrix $A \in \mathbb{R}^{n\times m}$ is represented by $\mathrm{Nz} \in \mathbb{N}$ the maximal number of elements unequal to zero per row, a representation matrix $\mathrm{A} \in \mathbb{R}^{n\times \mathrm{Nz}}$ for the elements unequal to zero, a representation matrix $\mathrm{j} \in {\mathbb{N}_0}^{n\times \mathrm{Nz}}$ for the indices of the elements and an information vector $\mathrm{rl} \in {\mathbb{N}_0}^{n}$ containing the number of the elements per row.
For

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 0 & 1 & 1 \\ 4 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix} \in \mathbb{R}^{4\times 3}$$

with $\mathrm{Nz} = 2$ we have the following ELLPACK-R representation:

$$\mathrm{A} = \begin{pmatrix} 1 & 3 \\ 1 & 1 \\ 4 & * \\ 2 & * \end{pmatrix} \in \mathbb{R}^{4\times 2}, \mathrm{j} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \\ 0 & * \\ 2 & * \end{pmatrix} \in {\mathbb{N}_0}^{4\times 2}, \mathrm{rl} = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \end{pmatrix} \in {\mathbb{N}_0}^{4},$$

which has to be saved in column-major order. The $(*)$-elements are replaced with zeros. This representation is dense in many applications (e.g. discretized surfaces).

5.3.2 Matrix-Vector Multiplication

Let A be a matrix in ELLPACK-R representation. We realize the matrix-vector multiplication with n threads as seen in Algorithm 4.

Input: $A \in \mathbb{R}^{n\times m}$, $v \in \mathbb{R}^m$
Output: $u = Av \in \mathbb{R}^n$

for $threadIndex \leftarrow x \leftarrow 0$ **to** $n-1$ **do in parallel**
  $svalue \leftarrow 0$
  $max \leftarrow \mathrm{rl}[x]$
  for $i \leftarrow 0$ **to** $max - 1$ **do**
    $value \leftarrow \mathrm{A}[x+in]$
    $col \leftarrow \mathrm{j}[x+in]$
    $svalue \leftarrow svalue + value \cdot \mathrm{v}[col]$
  $\mathrm{u}[x] \leftarrow svalue$

Algorithm 4: Matrix-vector multiplication in ELLPACK-R format.

Let t be the number of threads which run in parallel. By using the ELLPACK-R format the runtime can be decreased from $\mathcal{O}(nm)$ to $\mathcal{O}(nNz/t)$.

6 Running Time

In this paper the cg-algorithm was parallelized with the described methods. It terminates after m_{max} iterations, if the 2-norm of the residual is less than a given upper bound of the error. By using the upper estimates for the parallelized operations we get a runtime in

$$\mathcal{O}\left(\frac{m_{max}}{t} n(Nz + \log n)\right),$$

in which n is the dimension of the coefficient matrix with at most Nz elements unequal to zero per row and t denotes the threads running in parallel. This promises a significant acceleration compared to the runtime of the sequential implementation, which is in $\mathcal{O}\left(m_{max} n^2\right)$.

6.1 Runtimes of the Algorithm

This section provides a comparison of the runtimes of the sequential and the parallel implementation of the conjugate gradient algorithm. As an application the heat equation is solved and different sizes of the $n \times n$-coefficient matrix are chose. When solving a linear system the convergence speed of the conjugate gradient algorithm to compute an acceptably accurate solution strongly depends on the condition number of the system matrix. Therefore, the time per iteration is used for detailed comparison. The table below presents this times (in ms) of different CPU implementations (Armadillo, MATLAB, MKL), the runtimes of the presented GPU version in CUDA 3.1 (highlighted in orange) and a CUDA 3.2 implementation from NVIDIA.

n	CPU	Matl.	MKL	Cu. 3.1	Cu. 3.2
512^2	12.9	13.3	3.4	3.1	3.0
1024^2	56.6	40.6	19.4	7.3	4.6
2048^2	238.6	153.0	79.9	21.5	9.2

The second column (CPU) contains the runtimes of a sequential CPU implementation. This implementation uses the Armadillo C++ Library 1.0.0 (based on LAPACK) for scaled vector additions and dot product calculations. The matrix is stored in the ELLPACK-R format to realize an efficient matrix-vector multiplication.
Intel®MKL 10.3 (Math Kernel Library) is a library of threaded math routines, which contains an implementation of the conjugate gradient algorithm using the compressed row storage (CRS) format to store the matrix. The convergence tolerance from the correct solution is set to 1.0 regarding the 2-norm of the residual. The implementations were tested on a PC with Intel® Core™ i7 860 (2.80 Ghz) and NVIDIA® GeForce® GTX 480 graphics card. In the CUDA 3.1 implementation the number of threads per block was set to 512, because on the used hardware the configuration was always optimal. This is shown in Figure 4 in case of the $n = 512^2$ matrix.
For the implementation of the conjugate gradient solver, which is described in this paper the CUDA Toolkit 3.1 was used. NVIDIA has now published the version 3.2, which includes a conjugate gradient solver using the CRS format. This solver is in case of large matrices about a factor of two faster then the described implementation.
Additionally it should be mentioned that the data transfer

time to the GPU memory is compensated when using coefficient matrices with dimension $n > 1000$.
Collectively, we can say that in comparison to the sequential CPU implementation (MATLAB) a speed up between 4 to 7 is reached with the presented implementation for the tested coefficient matrices depending on the size. For larger coefficient matrices additional speed up is expected.

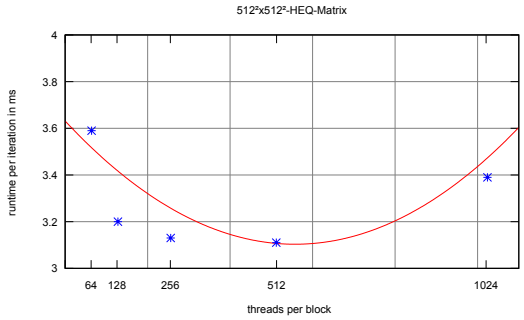


Figure 4: Dependence of the runtime of the number of threads per block.

7 Application: Heat Equation

In this paper the parallel cg-algorithm was tested on solving the heat equation. Let G be a $n \times m$ grid (see Figure 5) and let $u(x,y,t)$ be the temperature at a given time t at the point $(x,y) \in G$. The temperature gradation on the grid is in case of diffusion represented by the differential equation

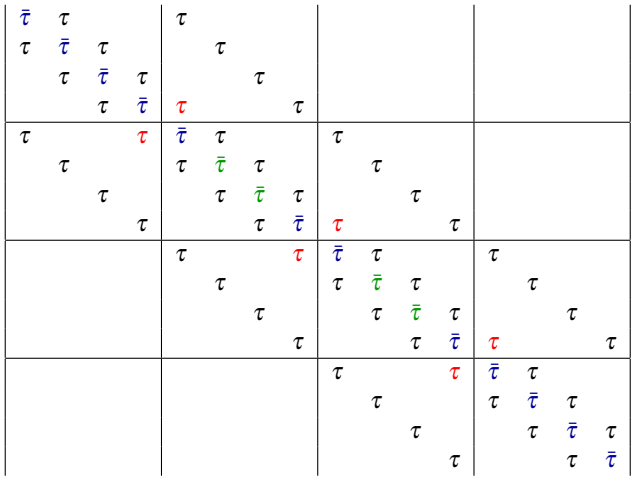
$$\frac{\partial u(x,y,t)}{\partial t} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2}.$$

7.1 Discrete Heat Equation

Using finite differences the discretization of the heat equation is given by

$$\begin{aligned} u(x,y,t) = (1-4\tau)\ & u(x,y,t+\Delta t) \\ +\tau\ & u(x+\Delta x,y,t+\Delta t) \\ +\tau\ & u(x-\Delta x,y,t+\Delta t) \\ +\tau\ & u(x,y+\Delta y,t+\Delta t) \\ +\tau\ & u(x,y-\Delta y,t+\Delta t) \end{aligned}$$

in the case of $\Delta x = \Delta y$ and $\tau := -\Delta t/\Delta x^2$. A problem occurs if a discretized point (x,y) is located on the boundary of the grid, because such a point is located in a neighborhood consisting of only two or three points. In such a case the term for the missing grid point has to be removed. The equation above can then be written as a linear equation system and is shown in Figure 5 for $n = m = 4$ with $\bar{\tau} := 1 - 4\tau$. The coefficient matrix of the system $A = A(G)$ can be represented as



In this case the red elements in the cofficient matrix cause a wrap around from the and the left-hand side and will be removed.

For larger grid sizes the coefficient matrices can be derived analogously. From the previous statements a discretization of the heat equation can be done using finite differences. The computation of the heat dissipation is done by solving a linear system of size $n \cdot m$ (cf. Figure 5) starting from a given initial configuration $u(0) \in \mathbb{R}^{n \cdot m}$. The resulting coefficient matrix is symmetric and positive definite, wherefore the solution can be computed using the conjugate gradient algorithm. From a current heat dissipation the previous dissipation can be computed by multiplying the coefficient matrix with the current result.

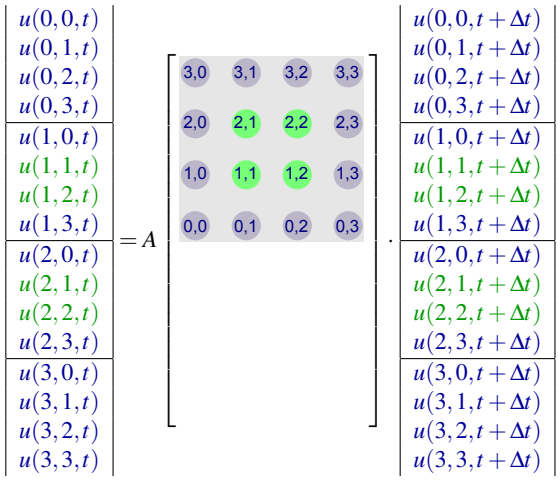


Figure 5: Discretization of a 4×4-grid.

7.2 Implementation

The presented parallel conjugate gradient algorithm was used to solve the 2D heat equation (see Figure 6). The implementation was tested (hardware specification above) with square images with different sizes 256, 512 and 1024

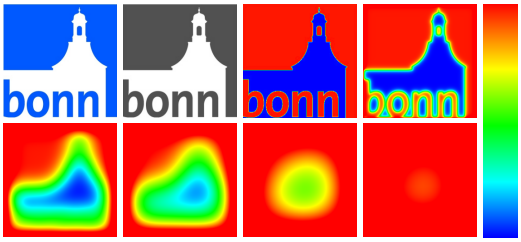


Figure 6: Heat distribution on the 256×256-emblem of the University of Bonn.

resulting in simulating the heat flow in 20, 10 and 7 time steps per second (interactive frame rates).

8 Future Work

In this paper, an accelerated cg-algorithm was implemented. To reduce the processing time the coefficient matrix was stored in ELLPACK-R format and the matrix vector operations were parallelized.
Another approach to advance stability and acceleration is the preconditioning of the matrix. A sequential implementation of the preconditioned conjugate gradient algorithm (pcg) was implemented in MATLAB to evaluate this.
Furthermore, the use of Shared Memory and Texture Cache could accelerate the runtime on older graphics cards.

8.1 Precondition and SSOR

One approach to advance stability and acceleration is the preconditioning of the coefficient matrix A. The cg-algorithm converged much faster for coefficient matrices with smaller condition number. For example this is shown in [12].
We can use this by left multiplication of the matrix $A \in \mathbb{R}^{n\times n}$ with a preconditioning matrix M^{-1}, in which $M \in \mathbb{R}^{n\times n}$ is symmetric and positive definite. We have to choose M in a way, where $\kappa(M^{-1}A) << \kappa(A)$. So we can solve the equivalent linear system $M^{-1}Ax = M^{-1}b$ instead of $Ax = b$. But a problem occurs, if we want to use the cg-algorithm to solve it, because the new coefficient matrix $M^{-1}A$ is in general not symmetric and positive definite. So we have to decompose the matrix with the Cholesky method and get $M = M_1M_2$ called the left and the right preconditioning matrix and execute the cg-algorithm on the linear system

$$\underbrace{M_1^{-1}AM_2^{-1}}_{=:\widetilde{A}} M_2x = \underbrace{M_1^{-1}b}_{\widetilde{b}}.$$

That is possible, because $\widetilde{A}$ is symmetric and positive definite. So we can solve the linear system $Ax = b$ in two steps. First we have to solve $\widetilde{A}y = \widetilde{b}$ with the cg-method. After that we can get the solution x by solving $M_2x = y$.
The second step is realizable with back substitution, because M_2 is an upper triangular matrix.
If we do it that way we have to calculate the matrix product $\widetilde{A} = M_1^{-1}AM_2^{-1}$ first. This would be a kind of a bottleneck. To avoid this we can use the substitution $\hat{x} = M_2x$, $\hat{r} = M_1^{-1}r$ and $\hat{p} = M_2P$ in Algorithm 2. This way we get the simplified preconditioned conjugate gradient algorithm (see Algorithm 5, c.f. [11]).

$x^{(0)} \leftarrow 0$
$r^{(0)} \leftarrow b$
$p^{(0)} \leftarrow M^{-1}r^{(0)}$

for $m \leftarrow 1$ **to** n **do**
  $\alpha^{(m)} \leftarrow \left\langle r^{(m-1)}, M^{-1}r^{(m-1)} \right\rangle / \left\langle p^{(m-1)}, Ap^{(m-1)} \right\rangle$
  $x^{(m)} \leftarrow x^{(m-1)} + \alpha^{(m)}p^{(m-1)}$
  $r^{(m)} \leftarrow r^{(m-1)} - \alpha^{(m)}Ap^{(m-1)}$
  $\beta^{(m)} \leftarrow \left\langle r^{(m)}, M^{-1}r^{(m)} \right\rangle / \left\langle r^{(m-1)}, M^{-1}r^{(m-1)} \right\rangle$
  $p^{(m)} \leftarrow M^{-1}r^{(m)} + \beta^{(m)}p^{(m-1)}$
return $x^{(n)}$

Algorithm 5: Preconditioned conjugate gradient (pcg).

In this paper the preconditioned algorithm was tested with the SSOR method (Symmetric Successive Overrelaxation, cf. [3]) in MATLAB. This method uses the strict lower triangular matrix L and the diagonal matrix D of A.
The SSOR preconditioning matrix

$$M := C^{-1} = (D+L)D^{-1}(D+L)^t$$

is an approximation of the coefficient matrix A.
In every iteration we have to solve the linear system

$$C^{-1}z = r^{m-1}$$

to get $z = Cr^{m-1} = M^{-1}r^{m-1}$.
In the MATLAB implementation the decomposition $C^{-1} = KK^t$ with $K = (D+L)D^{-1/2}$ was used to solve the system with forward and back substitution.
A disadvantage of SSOR is the use of $D^{-1/2}$, which is only available for matrices with a positive main diagonal. All elements of $D+L$ are included in A. So the required memory is less.
The implementation shows a convergence in mind of the number of required iterations to get a precise approximation of the solution, which is significant faster compared to the unpreconditioned method. But the algorithm solves an linear system in every iteration, for which reason the total runtime is only faster for large ($n > 1000$) and bad conditioned matrices. Another aspect is the enhanced stability because of the decreased condition number.
This concept could be another approach to accelerate the parallelized algorithm. For this purpose it is essential to realize the solution of the linear system $Mz = r^{m-1}$ efficiently, for example with parallelization. The parallelization of the back substitution is still an unsolved topic in science.

8.2 Shared Memory and Texture Cache

In the implementation, which is described in this paper the input data were copied into the global memory of the graphics card. During the execution, the threads get the required data only from this memory, which is about 512 to 2048 MB on modern graphics hardware. There exists also a comparatively small shared memory (it can also be configured as a L1-cache) and texture cache, which is only about a few KB. The access time to this kind of memory is much shorter (sometimes about a factor in the dimensions of 100) in comparison to the often uncached global memory (cf. [13]). Because of the small size in the most cases it is not possible to store the whole coefficient matrix inside of it.
Another way to speed up the process is the skillful use of this memory. Therefore, it is necessary to copy parts of data for future calculations from global memory to shared memory and texture cache. In order to achieve an acceleration, this must be constructed so that most of these transfers occur in parallel to the running threads.

9 Conclusion

This work presents a parallel implementation of the conjugate gradient algorithm using the NVIDIA CUDA architecture. The operations which were parallelized are the scaled vector addition, the dot product and the sparse matrix-vector multiplication. As sparse matrix format ELLPACK-R was used which provides an memory efficient storage for large coefficient matrices on the GPU. To compare the presented implementation, a comparison with a sequential CPU implementation using the LAPACK-based Armadillo C++ Library 1.0.0 and different CPU implementations (MATLAB, MKL) was made. In comparison to the sequential CPU implementations the parallel version of the conjugate gradient algorithm is in average 4 to 7 times faster. The speed-up of the algorithm is further increased if larger coefficient matrices are used. The transfer time between the CPU and the GPU will be compensated if the system matrices are acceptably large, in the presented case $n > 1000$. An additional speed up can further be gained if a preconditioner is used (e.g. SSOR). In this case in each iteration an additional equation system must be solved by back subsitution but currently there exists no efficient solution to that problem.

References

[1] G. Alefeld, I. Lenhardt, and H. Obermaier. *Parallele numerische Verfahren*. Springer, New York, Berlin, Heidelberg, 2002.

[2] M. Ament, G. Knittel, D. Weiskopf, and W. Strasser. A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, 0:583–592, 2010.

[3] M. Ament, G. Knittel, D. Weiskopf, and W. Strasser. A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP '10, pages 583–592, Washington, DC, USA, 2010. IEEE Computer Society.

[4] N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec. 2008.

[5] J. Bolz, I. Farmer, E. Grinspun, and P. Schrï¿½der. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM TRANSACTIONS ON GRAPHICS*, 22:917–924, 2003.

[6] J. Georgii and R. Westermann. A multigrid framework for real-time simulation of deformable volumes. In *Proceedings of the 2nd Workshop On Virtual Reality Interaction and Physical Simulation*, pages 50–57, 2005.

[7] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[8] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin. Feature based terrain generation using diffusion equation. *Computer Graphics Forum*, 29(7):2179–2186, 2010.

[9] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler. A local/global approach to mesh parameterization. In *Proceedings of the Symposium on Geometry Processing*, SGP '08, pages 1495–1504, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[10] NVIDIA-Corporation. Nvidia cuda c programming guide, Version 3.1, 2010.

[11] Y. Saad. *Iterative methods for sparse linear systems*. Second edition, 2003.

[12] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1997.

[13] F. Vazquez, E. M. Garzon, J. Martinez, and J. J. Fernandez. The sparse matrix vector product on gpus. *aceuales*, pages 1–13, 2009.

Physical Animation of Wetting Terrain and Erosion

Matej Hudak*
Supervised by: Doc. RNDr. Roman Durikovic, PhD†

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

Abstract

Visual simulation of natural erosion on terrains with granular matter, like sand, soil or gravel has been a fascinating research topic in the field of computer graphics for a long time. Ability of a fluid to drastically change internal structure or external shape of terrain is an important effect in nature. While there are many particle based algorithms to improve process of terrain erosion, only few of them also take into account the water saturated in a terrain.

In this paper we present a particle-based method for large scale long time progressive simulation of terrain erosion containing wet granular particles. The wetting process and the propagation through granular material is based on defining the wetness value for each particle representing the amount of water absorbed by granular particles and stored between them, as was originally proposed by Rungjiratananon [12]. We extend this model by adding a non homogeneous material to simulate differences between different types of soil-like granular material, based on physical constants like stability, plasticity and wetness. With this approach we can create a physical animation of erosion process like mass movement or mass wasting.

Keywords: Physical Animation, Particle-based Simulation, Erosion of Terrain, Mass Movement

1 Introduction

The terrain erosion is important process in the nature. Granular materials like sand or soil and their behavior due to influence of erosion is indispensable part of modeling naturally looking terrain. We can represent granular materials as large ensembles of particles, where in case of soil materials each element is non deformable [19]. One of the most important factors acting on particles of granular materials is water. Influence of this element changes shape, morphology and properties of material with result in erosion.

In this paper we present particle-based simulation method of water influence when applied to granular materials resultant in erosion. Although there are many algorithms to simulate such behavior, majority of them is acting on the terrain surface. Our algorithm take into account also water saturated in material. In natural erosion, certain amounts of water are creating wetness and spreading among little gaps between granular particles. Result of wetness propagation and its gathering between different layers of soil-like material is the erosion like mass movement, earth flow or slump.

Figure 1: Example of mass movement erosion.

We define soil system in this paper as large structure of granular rigid particles with values of stability, wetness and friction. Shape of particles is spherical, as in huge masses of particles, naturally behaving friction, between them, is inefficient [15]. For purpose of simulating differences between different types of soil in real world we created layers of soil material, where each layer represents one type of soil in the real world. Layers are bounded by forces acting on them and between them.

Similarly to general erosion, mass movement can be also described as three-step erosion process [23]. In the first step the regolith or boundary between layers of soil is damaged by influence of gathered wetness between layers and other factors such as physical structure of soil layers. During this process, water interacts with terrain by bringing wetness to its structure. The time required to create such failure ranges from few weeks to few years. In the second step, corrupted mass of eroded material is transported by natural factors such as gravity, weight of wet soil and shape of stable, not moving terrain. Last step of erosion involves a deposition of eroded material. At this stage of simulation, deposition of material is considered in large scale.

*subseth.mato@gmail.com
†durikovic@fmph.uniba.sk

For a simulation of soil material we use *Discrete Element Method* (*DEM*), which comprises different techniques suitable for simulating dynamic systems behavior with multiple rigid and separated bodies of various shapes. Continuous changes, computed in contact forces and applied in contact status, turn influence of the subsequent movement of particles [17]. Since positions of particles are changed by physical forces designed in the contact states of particles, topology of particle interaction evolves freely. As a result, highly dynamic simulations, such as avalanches and general erosion can be conveniently generated by this meshless approach without sacrificing physical accuracy [4]. *Smoothed Particle Hydrodynamics* (*SPH*) [18, 16] is used to simulate water particles.

This paper is organized as follows. In the section 2, we present algorithms, which are most closely related to our work. This is followed by section 3.1, where we describe our algorithm for soil simulation system and erosion. Thereafter, in section 5, we describe visualization method and optimization of given algorithm. Finally, in Section 4 and 3.4 we present mass movement simulation and results of our algorithm. The paper concludes with conclusions and future work.

2 Related Work

Erosion

The simulation of terrain erosion was interesting area of research in a Computer Graphics for a long time. One of first representations for terrains started with mathematician, Benoit Mandelbrot [3], father of fractal geometry, who introduced using fractals in terrain modeling. One of the first algorithms [11] applied thermal and hydraulic erosion to erode fractal terrains. In the topic of generating differently shaped fractal mountains [21] authors used water as main factor in simulation. Layered structure and its application in thermal weathering was introduced by Benes et al. [23]. The authors used layers of terrain, water and dissolved material. Transport of material in terrain erosion was described in [5], where erosion model uses also cohesive force between particles. Interactive simulation of erosion using water as main factor for creating changes on the surface of terrain was presented in [13]. Quite recently terrain erosion simulated with *SPH* was presented in [10]. Authors used Smoothed Particle Hydrodynamics to dissolve some amount of material from ground, transported due to water which created deposition of material on a different place.

Granular Materials

One of the first attempts to simulate granular material was introduced by Cundall [17], who described Discrete Element Method for simulating rocks mechanics, based on his earlier works [1, 2]. Granular material like sand was well described by many articles. Some of them used height field methods for better performance of simulation [20, 14, 25] or handle the material as fluid [24]. Although these methods are quite efficient, they are less accurate and difficult to use for more complex simulation system. Idea of using *DEMs* for simulation of granular material was revisited by Bell et al. [4], who also described different types of friction and created non spherical particles to demonstrate real friction force in simulation. Recently wetness in sand material was introduced with *DEM* and *SPH* method [12, 22]. For simulation of realistic static friction [15] used counter-acting frictional force. They also showed that piles generated by avalanches have finite angle of repose. In the study of [7, 8, 9, 6] authors introduced approach to 3D simulation of cohesive and non cohesive system. To simulate cohesion in soil system they used *DEM* and bonding forces between particles.

3 DEM for Soil

3.1 Discrete Element Method

At first step of our algorithm, we define *Discrete Element Method* for simulating granular material such as soil. Sand is one of soil types, with diameter larger than $0.02\,mm$. This is first type of material in our simulation. It is well described and simply modeled in particle-based simulations, thus it is starting material in simulation of erosion.

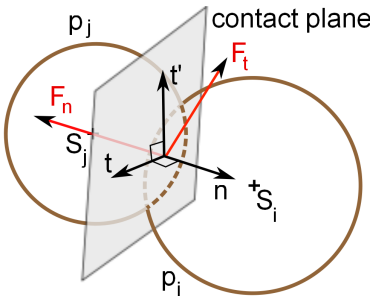


Figure 2: Contact forces for *DEM* method.

Contact forces between two colliding particles p_i and p_j, introduced by *Cundall* [17], are constructed acting in contact point between particles. In the figure 2, we can see basic setup scene for contact forces in *DEM*. At first we define overlap value of colliding particles and normal:

$$\xi = max(0, r_i + r_j - ||\overrightarrow{x_i} - \overrightarrow{x_j}||), \tag{1}$$

$$\overrightarrow{N} = \frac{\overrightarrow{x_i} - \overrightarrow{x_j}}{||\overrightarrow{x_i} - \overrightarrow{x_j}||}, \tag{2}$$

where $\overrightarrow{x_i}$ and $\overrightarrow{x_j}$ are positions of particles p_i, p_j and r_i and r_j are radiuses of these particles. Following equations describe normal force and its computation between particles p_i and p_j.

$$\overrightarrow{F_n} = \overrightarrow{F_s} + \overrightarrow{F_d}, \tag{3}$$

$$\overrightarrow{F_s} = k_s \xi \frac{\overrightarrow{N}}{||\overrightarrow{N}||}, \tag{4}$$

$$\overrightarrow{F_d} = k_d \overrightarrow{N}, \tag{5}$$

where $\overrightarrow{F_s}, \overrightarrow{F_d}$ is spring and damping force and k_s, k_d is spring and damping coefficient. Coefficient k_s is determined and k_d is selected same or smaller. After positive overlap, normal force is applied to accelerations of particles p_i and p_j. In summation through all accelerations of particle p_i, we compute its velocity using Newton's second motion law and determine its new position after one step in time Δt.

For simulating natural granular material we are using friction applied in tangent direction. Friction or tangential force causes negative contribution to summation in generating particle's acceleration. We implemented basic types of friction forces described by Bell et al. [4]. We also implemented counter-acting friction force to simulate natural friction [15]. Unfortunately, to update this force, we need to hold list of old and new neighbors in system, which is very inefficient. Moreover, this force has minimal effect to more stable friction between particles. With this reasons, we decided to use following friction force

$$\overrightarrow{F_t} = -min(\mu f_n, k_t ||\overrightarrow{V_t}||) \frac{\overrightarrow{V_t}}{||\overrightarrow{V_t}||}, \tag{6}$$

where $\overrightarrow{V_t}$ is tangential velocity, which is tangent to the contact plane and perpendicular to the normal direction. The tangential velocity is defined using the relative velocity of the particles p_i and p_j at the contact point. Coefficient of friction k_t is limited by Coulomb law of friction, where $\mu$is friction coefficient and f_n is value of applied normal force $\overrightarrow{F_n}$. In figure 3 we can see simulated sand with *DEM* in our algorithm.

Figure 3: Sand simulated with our algorithm. Dry feature of granular material.

3.2 Strength of Material

Of course, sand-like material is generally too simple for erosion of soil causing mass movement. We need material which will be acting like stable structure with range of strength simulating natural soil. Forces applied in *DEM* during sand simulation are limited. For simulating dry soil we need to introduce another force to preserve strength of dry material constructed from rigid particles.

For that purpose we implemented bonds to our algorithm. Bond is relation between two colliding particles in case when they are in relax state or overlapping [9]. This force is applied to basic definition of *DEM* in the meaning that *normal bond* is applied to *normal force* $\overrightarrow{F_n}$ etc. In figure 4 we can see setup example of two disks representing two particles p_i and p_j.

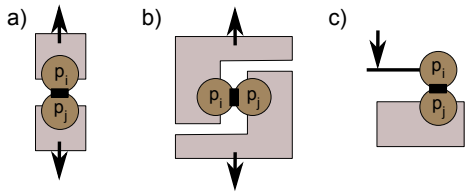


Figure 4: Bonded motion of pair of particles. Black squares are representing bonds. Arrows are pushing particles away from each other in (a) normal, (b) tangential and (c) angular direction. Bonds interlock particles and keep them together.

In example, for normal force, if $\overrightarrow{F_n}$ in (3) is resolving collision of particles by pushing them away from each other, than bonding force in normal direction (7) acts conversely. We can define normal bonding force as follows

$$\overrightarrow{B_n} = (R_n/\xi) * \overrightarrow{N}, \tag{7}$$

where R_n is normal bonding coefficient and ξ is overlap value between particles. Tangential and angular motion are locked in similar way using tangential and angular bonding coefficients R_t and R_ω. After considering this force, we need to update all equations in *DEM* contact model by subtraction of these bonding forces. Bonds are limited by spring and tangential forces acting in *DEM*. It means that failure of system is not allowed. In our algorithm, we are using diameter value of particles.

In summation from above equations we can see, that bonds create forces, which hold particles together in normal, tangential and angular direction, if they are overlapping. In the same time rigidness of granular particle in simulation is preserved.

In case of sand, rotation of particles can be neglected, because particles can be easily separated during time of simulation. In case of such complex and complicated process as terrain erosion we must also consider rotations of bigger masses of soil constructed with certain amounts of particles. Thus in our algorithm rotation can not be ignored. To include angular motion of particle we compute angular acceleration of colliding particles. Then in integration of step of time Δt we determine particle's angular velocity. Rotation is applied to particle's body using rotation matrix, computed using quaternions. Then in next time step relative velocity $\overrightarrow{v_{pi}}$ of particle is simply updated to consider also angular velocity

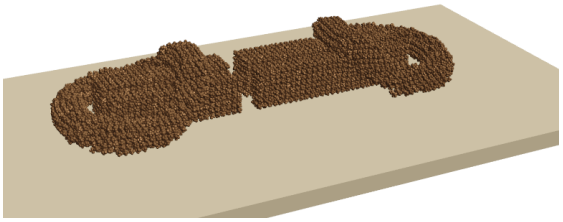


Figure 5: Dry feature of soil-like material with defined normal, tangential and angular bonds.

$$\overrightarrow{v_{p_i}} = \overrightarrow{v_{p_i}} + \omega_{p_i} \times (\overrightarrow{x} - \overrightarrow{p_i}), \tag{8}$$

where ω_{p_i} is angular velocity of particle p_i, $\overrightarrow{x}$ is position of contact and $\overrightarrow{p_i}$ is position of particle p_i.

3.3 Wetness

Water is main factor to cause mass movement. It is water absorbed in inner soil structure, which causes bigger weight of wetted soil and then after some time, failure state of system, when mass movement starts. With this condition we implemented wetting system to our granular soil-like material.

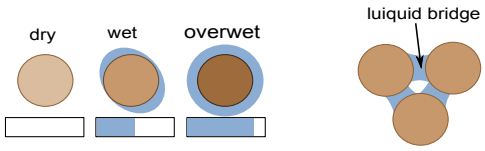


Figure 6: States of wetness in particles (left). Liquid bridges (right).

This part of algorithm is based on wetness system presented by Rungjiratananon et al. [12]. In our algorithm there are also three states of particles, dry, wet and overwet as we can see in figure 6. Wetness is then percentage expression of each state, applied to particle. This percentage value represents water saporated in little gaps between particles.

Amounts of water between two particles represented by wetness, create attractive acting force, which can simulate cohesion of material. This force is acting between particles in case they are moving away from each other

$$\overrightarrow{F_i^{attract}} = max\left\{0, w_f - \frac{w_i - w_j}{2}\right\}(v_j - v_i), \tag{9}$$

where w_f is fluidization coefficient. As we can see, wetness system has impact on *DEM* contact model. Contact forces are updated similar to approach in Rungjiratananon's article [12]. With this feature, system becomes more plastic during loading wetness to its structure. Wetness between particles of material is propagated through

Figure 7: Example of wetness system with different materials using our algorithm.

material and controlled by coefficient of propagation k_p. In the layer with bigger strength, propagation coefficient is smaller. Wetness is propagated to all contacts N_i of particle p_i as follows

$$w_i^{t+\Delta t} = w_i^t + k_p \frac{\Delta w_i^t}{N_i} \Delta t, \tag{10}$$

$$\Delta w_i^t = w_i^t + w_t, \tag{11}$$

where Δw_i^t is excessive wetness of particle p_i. As propagation speed of wetness is different in different layers of soil, excessive wetness is most visible on boundaries between layers of different material. These regions in materials are very hazardous because their behavior is water-like and they are creating chance for bigger mass of material to start a mass movement.

3.4 Summation of Algorithm

In figure 8 we can see diagram of accumulation forces. *SPH*, *DEM*, *Bonds* and *Wetness* are here different methods used for computing forces between colliding particles.

4 Mass Movement

In the figure 9, we can see illustration of mass movement erosion, where surface of rupture is region of soil particles with most excessive wetness. This region is forced to behave like mud due to forces in overwetted particles. Wetness between layers creates slide. Then slump block is volume of soil above surface of rupture. This volume is transported during mass movement erosion and it's basic result from this kind of erosion. There are many other products of mass movement. Most closely related to change of shape of terrain is production of scarps. Mass movement

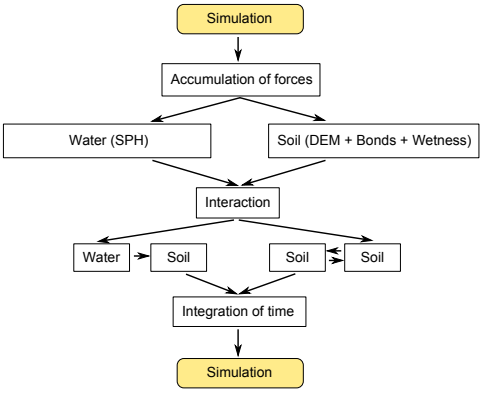


Figure 8: Accumulation of forces between particles.

can be very fast, or very depending on gradient, shape of terrain, amount of wetness, weight of wetted particles and also construction of initial layers.

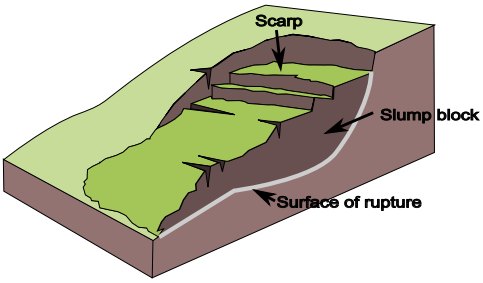


Figure 9: Mass movement illustration.

Simulation starts with setup input scene constructed from particles of soil and water. Model of terrain is created in *Blender* and saved in *.obj* format. With our application *VolumetoParticles*, we scan input model and create representation of particles using 3D scanline with defined positions of particles of soil with density and contact radiuses. Layers of terrain are presented with different objects in model. With this approach we can easily create synthetic input scene representing terrain as we can see in figure 10.

As initialization to our algorithm we setup starting scene. Then wetting of terrain can start. Particles of water simulate rain and bring required wetness to system of soil. Wetness is propagating through particles of soil to lower layers of material.

Each layer has different properties of strength, density and speed of wetness propagation. With combination of different layers in scene we can create non homogeneous material. Even each particle can have different initial properties and also predefined wetness. With increasing wetness within particles, weight is also increasing. After certain time of simulation wetness gathered on boundaries of layers is creating a slide action. In this process forces between couple of particles are corrupted and they can slide over each other. This action is essential in our simulation of mass movement.

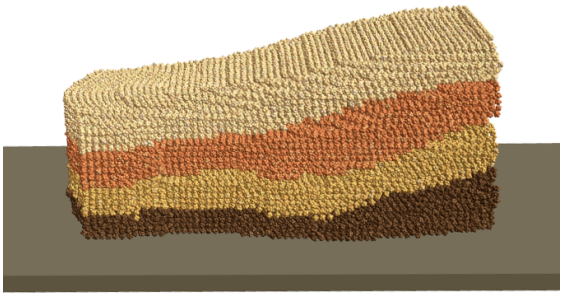


Figure 10: Input terrain. Layers are orderd from top to bottom in the meaning of water propagation speed.

scene	particles	1 core	openMP	time step
sand	130k	0.2 fps	6 fps	0.001
wetness	20k	7 fps	13 fps	0.0005
input	50k	4 fps	5 fps	0.0002
hill	30k	2 fps	3 fps	0.0001
layers	60k	1.2 fps	3.4 fps	0.0002

Table 1: Comparison of frames per second on different terrains without and with openMP.

5 Visualization and Optimization

For visualization of results of our work we are using *OpenGL*. Without lose of resolution in simulation it is not possible to simulate this type of erosion in real time using *DEM* method. As differential equations solver, we implemented $Runge-Kutta\,fourth-order$ algorithm with total accumulated error order h^4 and basic *Euler* algorithm.

Because of optimization we joined particles of water and soil to one programmable structure. They have different properties and they are simulated with different methods. With this feature we can compute contacts between particles more effectively. With assumption of using particles of water just to bring required wetness to soil system, we do not need to create surface of water and visualization. Physical correctness of particles of water is preserved.

For optimization of performance we used *openMP* for simulation on more threads of *CPU*. As hardware for simulation we used Intel i7 950 CPU with 8 cores. In following table, you can see performance of our algorithm and comparison with openMP optimization.

6 Results

We simulated random input terrains to test our algorithm. In figure 11 we can see wetting simulation performed on terrain with layers. In this terrain, there are 6 layers with different speed of wetness propagation. After close measurement, layer with red color is most resistant to wetness. As result, the layer, which is directly above most resistant one is layer with hugest amount of wetness between particles.

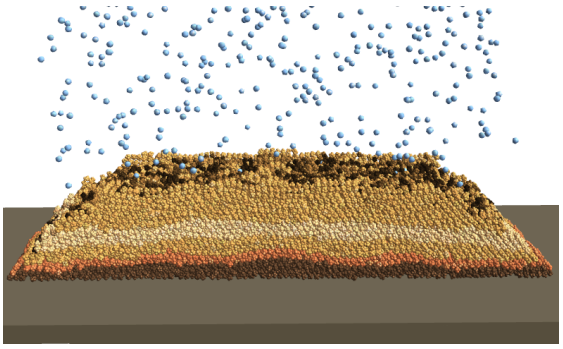


Figure 11: Wetting system on set of layers.

Figure 12 shows the mass movement simulation. Sample hill was constructed from three soil-like layers. Inspired by previous result, bottom layer is here the most resistant to wetness. As result red layer is capturing all incoming wetness and creates surface of rupture with result in movement.

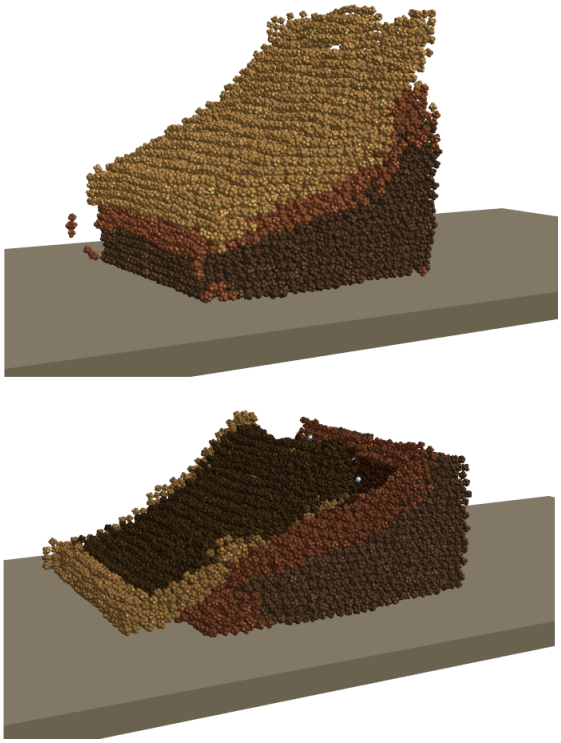


Figure 12: Mass movement on sample hill with 3 layers.

7 Conclusions and Future Work

In conclusion, we created particle based system for simulating soil particles and mass movement erosion. Non homogeneous material, layered data structure of soil, additional wetness, wetting and over wetting of material in our algorithm provide ideas for future work. With different layers of soil and interaction with water we are able to simulate formation of underlying structures such as caves and underlying water. With definition of high stability and strength of dry and also wet soil, it is possible to simulate drying of particles of soil. Although, described methods are not suitable for real time simulations, real time simulation of this processes is also our future goal.

8 Acknowledgment

The author wish to thank to Doc. RNDr. Roman Ďurikovič, PhD. for his support and the perfect leadership in this work. We are also thankfull to Mgr. Michal Chládek and Mgr. Juraj Onderik for help, support and developing simulation of water, which we are using to simulate water particles.

References

[1] Cundall P. A. A computer model for simulating progressive, large-scale movements in blocky rock systems. *Proc. Symp. Int. Soc. Rock Mech., nancy 2, number 8*, 1971.

[2] Cundall P. A. Ball-a program to model granular media using the distinct element method. *Technical Note, Advanced Technology Group, Dames & Moore*, 1978.

[3] Mandelbrot B. B. & Wheeler J. A. *The fractal geometry of nature*. 1983.

[4] Bell N. et al. Particle-based simulation of granular materials. *In SCA '05: Proceedings of the 2005 ACM SIGGRAPH, Eurographics symposium on Computer animation, p. 77-86*, 2005.

[5] Benes B. et al. Hydraulic erosion. *Computer Animation and Virtual Worlds 17, p. 99-108*, 2006.

[6] Delenne J-Y et al. Mechanical behaviour and failure of cohesive granular materials. *Int. J. Numer. Anal. Meth. Geomech. 28, p. 1577-1594*, 2004.

[7] Donze V. F. et al. Advances in discrete element method applied to soil, rock and concrete mechanics. 2008.

[8] Jian M. J. et al. Bond rolling resistance and its effect on yielding of bonded granulates by dem analyses. *Int. J. Numer. Anal. Meth. Geomech. 30, p. 723-761*, 2006.

[9] Jiang M. et al. A simple and efficient approach to capturing bonding effect in naturally microstructured sands by discrete element method. *Int. J. Numer. Meth. Engng. 69, p. 1158-1193*, 2007.

[10] Kristof P. et al. Hydraulic erosion using smoothed particle hydrodynamics. *EUROGRAPHICS, volume 28, number 2, p. 219-228*, 2009.

[11] Musgrave F. K. et al. The systhesis and rendering of eroded fractal terrains. *In SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics anf interactive techniques, ACM Press, p. 41-50*, 1989.

[12] Rungjiratananon W. et al. Real-time animation of sand-water interaction. *Pacific Graphics '08, Volume 27 , Number 7, p. 1887-1893*, 2008.

[13] Stava O. et al. Interactive terrain modeling using hydraulic erosion. *Eurographics / ACM SIGGRAPH Symposium on Computer Animation*, 2008.

[14] Summer R. W. et al. Animating sand, mud, and snow. *Computer Graphics Forum 18, volume 1, p. 17-26*, 1999.

[15] Lee J. & Herrmann H.J. Angle of repose and angle of marginal stability: Molecular dynamics of granular particles. *In SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, p. 965-972*, 1992.

[16] Gingold R. & Monaghan J. Smoothed particle hydrodynamics - theory and application to nonspherical stars. *Monthly Notices of the Royal Astronomical Society 181, p. 375 - 389*, 1977.

[17] Cundall P. A. & Strack O. D. L. A discrete numerical model for granular assemblies. *Geotechnique 29, p. 47-65*, 1979.

[18] Chladek M. *Flood simulations of city*. FMFI UK, Bratislava, 2010.

[19] Lal R. & Shukla K. M. *Pricinples of soil physics*. Marcel Dekker, INC., The Ohio State University Columbus, Ohio, New York, Basel, U.S.A., 2004.

[20] Li X. & Moshell J. M. Modeling soil: realtime dynamic models for soil slippage and manipulation. *In SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, p. 361-368*, 1993.

[21] Prusinkiewicz P. & Hammel M. A fractal model of mountains with rivers. *In Proceeding of Graphics Interface '93, p. 174-180*, 1993.

[22] Lenaerts T. & Dutre P. Mixing fluids and granular materials. *EUROGRAPHICS '09, Volume 28, Number 2, p. 213-218*, 2009.

[23] Benes B. & Forsbach R. Layered data representation for visual simulation of terrain erosion. *In SCCG '01: Proc. of the 17th Spring conference on Computer graphics (2001), volume 25, p. 80-86*, 2001.

[24] Zhu Y. & Bridson R. Animating sand as a fluid. *In SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, p. 965-972*, 2005.

[25] Onoue K. & Nishita T. Virtual sandbox. *In PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, p. 252*, 2003.

Fast Hydraulic and Thermal Erosion on the GPU

Balázs Jákó*
Supervised by: Balázs Tóth†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest/Hungary

Abstract

Computer games, TV series, movies, simulators, and many other computer graphics applications use external scenes where a realistic looking terrain is a vital part of the viewing experience. Creating such terrains is a challenging task. In this paper we propose a method that generates realistic virtual terrains by simulation of hydraulic and thermal erosion on a predefined height field terrain. The model is designed to be executed interactively on parallel architectures like graphics processors.

Keywords: Erosion, simulation, GPU, GPGPU, hydraulic, thermal.

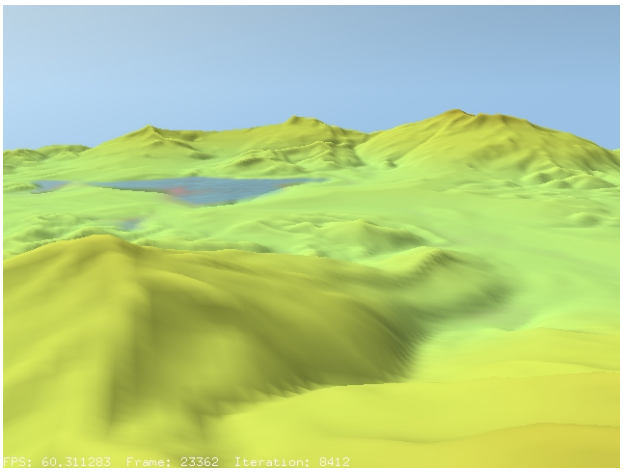


Figure 1: Landscape with mountains, valleys, ridges, riverbeds and lakes, generated by our method.

1 Introduction

Natural eroded terrains have some typical features like valleys, riverbeds, ridges, etc. These are results of different kinds of erosion caused by water, thermal shocks, and wind. Hydraulic erosion is caused by running water on terrain surface generated by from falling rain and springs. The flowing water dissolves soil and transports it to lower locations where the dissolved sediment is deposited. Thermal erosion is caused by temperature changes caused by the alternation of the hot Sun and cold night air. Hard surfaces are cracking up into smaller parts, the decomposed material is moving down to lower areas due to gravity.

There are different approaches to generate virtual terrains with features caused by these phenomena. Methods based on fractal techniques build a terrain that is similar to real-world mountain scenery, but these are isotropic, making ridge and valley generation difficult. Topographical methods use structural models to simulate water systems. These emphasize water flow, but mountain slopes are less natural in results. Physics based models simulate erosion factors and their effects on terrain surface like water flow, thermal shocks, and wind. In our method, we focus on hydraulic and thermal erosion, because these have the most impact on the terrain surface.

The programmability of modern GPUs makes it possible to execute not only graphics algorithms but a wide range of other, more generic tasks [11]. Using general purpose graphics processing units (GPGPUs) for simulation is obvious when the simulation algorithm can be excuted in parallel. The architecture of the GPUs execute appropriate algorithms much faster than traditional CPUs which makes such simulations able to run interactively, allowing direct observation and manual intervention during execution.

Physics based erosion methods apply some kind of fluid simulation. Chiba et al. [4] were the first to propose simulating valleys and ridges using particle systems. Beneš et al. [3] presented a model that uses the Navier-Stokes equations on a 3D regular grid simulating the erosion process. Neidhold et al. [5] used simplified Newtonian physics model for velocity computation on a 2D grid. Up to this, all of these models are computationally expensive, and due to data dependencies they are hard to execute on a parallel hardware. 2D Navier-Stokes equations were solved efficiently by Harris [6] and Wu [14]. Kass et al. [7] solve shallow water equations in their model, which was also used by Beneš et al. [1], to simulate erosion in real-time. Mei et al. [8] used this model in their simulation with employing virtual pipes introduced by O'brien et al. [10] that is the key to parallel execution. Our approach is partly

*balazs.jako@hun-digital.hu
†tbalazs@iit.bme.hu

based on [8]. To improve the realism of the simulation, we adapted the thermal erosion model of Beneš et al. [2] by applying the idea of virtual pipes and merging it into the improved hydraulic erosion algorithm.

2 Erosion Model

Our hydraulic erosion model is an improved version of the method introduced by Mei, Decaudin and Hu [8]. This model works with a 2D uniform grid and uses the following quantities in each (x,y) cell (see Figure 2):

- terrain height b,
- water height d,
- suspended sediment amount s,
- water outflow flux $\mathbf{f} = (f^L, f^R, f^T, f^B)$,
- velocity vector $\overrightarrow{\mathbf{v}}$.

These values are updated in each iteration. The simulation iteration consists of the following five steps:

1. Water incrementation due to rain or water sources.
2. Flow simulation using shallow-water model. Computation of the velocity field and water height changes.
3. Simulation of the erosion-deposition process.
4. Transportation of suspended sediment by the velocity field.
5. Water evaporation.

These steps are executed in each iteration step, gradually changing the state variables in each cell. Let $b_t, d_t, s_t, f_t, \overrightarrow{v_t}$ denote the data elements at a given time t and Δt the time step. In the following, we summarize the calculations producing the values at the next $t+\Delta t$ time. Since the model calculates some variable values in two or more steps, we will use subscripts $1,2,\ldots$ to distinguish the temporal values from the final output used in following iterations.

First, we simulate the effects of water arriving at the terrain surface. Unlike the original model, we use constant $r(x,y)$ rain rate for each cell instead of large randomly distributed raindrops falling down to surface. The rain rate specifies the water amount arriving at a given (x,y) cell during Δt time. This gives us more balanced and finer grained results in the long run. The water height is updated by the following formula:

$$d_1(x,y) = d_t(x,y) + \Delta t \cdot r_t(x,y) \cdot K_r \tag{1}$$

where K_r is a global simulation parameter that scales the overall rate of water increment, and d_1 is the intermediate value of the water height.

Then, we calculate the water flow between cells. Each (x,y) cell has four virtual pipes to the four neighbors which

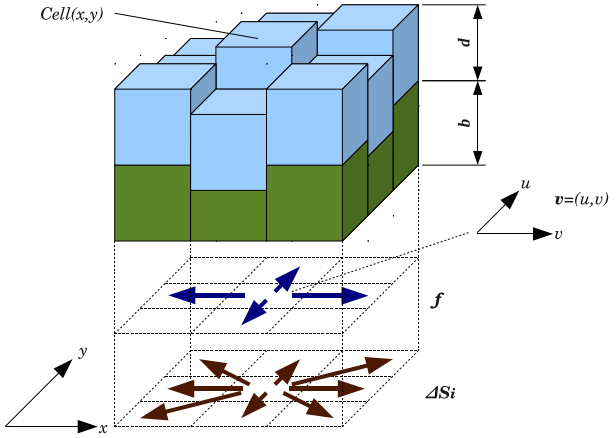


Figure 2: Water and thermal sediment flow model.

transport water outward from the given cell. Neighboring cells also have four virtual pipes, transporting water to opposite directions. The water outflow flux is updated with the pressure difference between interconnected cells. Let's denote $\mathbf{f} = (f^L, f^R, f^T, f^B)$ the outflow flux in a given (x,y) cell, where f^L is the outflow flux to the left neighbor at $(x-1,y)$, and similarly f^R, f^T, f^B are the outflow fluxes to right, top, bottom directions, respectively. We calculate the change of f^L as:

$$f^L_{t+\Delta t} = \max(0, f^L_t(x,y) + \Delta t \cdot A \frac{g \cdot \Delta h^L(x,y)}{l}) \tag{2}$$

where A is the cross section area of the virtual pipe, g is the gravity, l is the length of the virtual pipe, $\Delta h^L(x,y)$ is the height difference between the left and the current cell:

$$\Delta h^L(x,y) = b_t(x,y) + d_1(x,y) - b_t(x-1,y) - d_1(x-1,y) \tag{3}$$

The calculation of f^R, f^T, f^B is performed in a similar way. The total outflow should not exceed the total amount of the water in the given cell. If the calculated value is larger than the current amount in the given cell, then $\mathbf{f}$ will be scaled down with an appropriate K factor:

$$K = \max(1, \frac{d_1 \cdot l_x \cdot l_y}{(f^L + f^R + f^T + f^B) \cdot \Delta t}) \tag{4}$$

where l_x, l_y are the distances between the grid cells in the x,y directions. The outflow flux is multiplied by K:

$$f^i_{t+\Delta t}(x,y) = K \cdot f^i_{t+\Delta t}, i = L,R,T,B. \tag{5}$$

We calculate ΔV water height change with adding f_{out} output and f_{in} input flow values in each (x,y) cell:

$$\begin{aligned} \Delta V(x,y) &= \Delta t \cdot (\sum f_{in} - \sum f_{out}) = \\ &= \Delta t \cdot (f^R_{t+\Delta t}(x-1,y) + f^T_{t+\Delta t}(x,y-1) + \\ &\quad f^L_{t+\Delta t}(x+1,y) + f^B_{t+\Delta t}(x,y+1) - \\ &\quad \sum_{i=L,T,R,B} f^i_{t+\Delta t}(x,y). \end{aligned} \tag{6}$$

Then, we update the water height in the current (x,y) cell:

$$d_2 = d_2(x,y) + \frac{\Delta V(x,y)}{l_x l_y}. \tag{7}$$

Using outflow flux values, we can calculate the $\overrightarrow{\mathbf{v}}$ velocity field that needed to calculate hydraulic erosion and deposition. The calculation of the x component is:

$$\begin{aligned}\Delta W_x = \frac{1}{2}(f^R(x-1,y) - f^L(x,y) + \\ f^R(x,y) - f^L(x+1,y))\end{aligned} \tag{8}$$

The y component is calculated in a similar way. Since we know the velocity vector, we can calculate C water sediment transport capacity that represents how much sediment can be transported in a cell. In the original model C is calculated as:

$$C(x,y) = K_c \cdot \sin(\alpha(x,y)|\overrightarrow{\mathbf{v}}(x,y)|) \tag{9}$$

where K_c is a global simulation parameter controlling sediment capacity, $\sin(\alpha)$ is the local tilt angle, and $\overrightarrow{\mathbf{v}}(x,y)$ is the water flow vector in the cell. This empirical formula erodes terrain proportional to the surface slope. To allow some erosion at nearly flat areas, there is a lower limit for $\sin(\alpha(x,y))$. Fluid erosion in real-world is highly dependent on the water depth. Deep sea floors are practically never eroded, even if there is a stream in the water, because water flow is slower in deeper water levels, although sediment capacity of deeper water is larger due to the larger water volume. On the contrary, a relatively shallow river always dissolves the terrain at the bottom, because water is flowing faster and has direct effect on the bottom. To simulate this, we modified equation (9) by introducing l_{max} limiting function:

$$C(x,y) = K_c \cdot \sin(\alpha(x,y))|\overrightarrow{\mathbf{v}}(x,y)| \cdot l_{max}(d_1(x,y)) \tag{10}$$

$l_{max}(x)$ is a ramp function that is defined by the following:

$$l_{max}(x) = \begin{cases} 0, x \leq 0 \\ 1, x \geq K_{dmax} \\ 1 - (K_{dmax} - x)/K_{dmax}, 0 < x < K_{dmax} \end{cases}$$

where K_{dmax} is a global simulation parameter controlling the maximum erosion depth. This function scales down the fluid erosion effects by the water depth, so the erosion will occur only in shallower areas, forcing the simulation to dispose sediment at deeper water areas, just like in real world. This produces much more natural looking deep water sea and lake floors than the original model. Moreover, we included true 3D collision between water and terrain surface:

$$C(x,y) = K_c \cdot (-\overrightarrow{\mathbf{N}}(x,y) \cdot \overrightarrow{\mathbf{V}}) \cdot |\overrightarrow{\mathbf{v}}(x,y)| \cdot l_{max}(d_1(x,y)) \tag{11}$$

where $N(x,y)$ is the terrain surface normal at point (x,y) and $\overrightarrow{\mathbf{V}}$ is the 3D water flow vector calculated from the surface tangent and 2D velocity vector $\overrightarrow{\mathbf{v}}$. This modification erodes more soil if the water collides with the surface in angles closer to perpendicular. With our model, we observed some ripples on sea floors similar to sand ripples on real-world seashores.

At this point, there is a decision by using the C capacity. If transported sediment s_t in cell (x,y) is smaller than C, then we dissolve some soil in water:

$$b_{t+\Delta t} = b_t - \Delta t \cdot R_t(x,y) \cdot K_s(C - s_t), \tag{12a}$$
$$s_1 = s_t + \Delta t \cdot R_t(x,y) \cdot K_s(C - s_t), \tag{12b}$$
$$d_3 = d_2 + \Delta t \cdot R_t(x,y) \cdot K_s(C - s_t), \tag{12c}$$

where K_s is the global coefficient. Otherwise, if $C < s_t$, then we dispose some of the transported sediment in a similar way:

$$b_{t+\Delta t} = b_t + \Delta t \cdot K_d(s_t - C), \tag{13a}$$
$$s_1 = s_t - \Delta t \cdot K_d(s_t - C), \tag{13b}$$
$$d_3 = d_2 - \Delta t \cdot K_d(s_t - C), \tag{13c}$$

where K_d is a global parameter controlling deposition speed. Equations 12c and 13c were added to improve long-term stability. Originally, the $\Delta t \cdot K_d(s_t - C)$ suspended sediment amount was subtracted from the terrain heigth b_t without adding it to d_t water height. The overall height of the water surface is defined as $b_t + d_t$ at a given t time. Thus, after the subtraction the overall water surface height was decreased by the amount of the sediment suspended by the water itself, caused water to disappear with the sediment, ignoring the fact that suspended amount is still in the water in fluid form. This caused some unwanted feedback to the water flow simulation from the sedimentation process and causes regular ripples on the water surface in the long run. With our modification, this behavior can be eliminated.

To prevent negative water heights in equation 12c, we clamped the dissolved amount to water height in cell (x,y).

There is one more improvement in the model. In nature, moving sediment becomes softer by the time. To imitate this, we slowly lower the $R(x,y)$ local hardness coefficient of the terrain when some soil is disposed:

$$R_{t+\Delta t}(x,y) = max(R_{min}, R_t(x,y) - \Delta t \cdot K_h K_s(s_t - C)) \tag{14}$$

where R_{min} is the lower limit of hardness, K_h is a global coefficient controlling the sediment softening. The next step in the model is to move dissolved sediment along the water using $\overrightarrow{\mathbf{v}} = (u,v)$:

$$s_{t+\Delta t}(x,y) = s_1(x - u \cdot \Delta t, y - v \cdot \Delta t) \tag{15}$$

If point $(x - u \cdot \Delta t, y - v \cdot \Delta t)$ is not on the grid, the model uses linear interpolation between the four closest grid points. In the last step, we simulate water evaporation:

$$d_{t+\Delta t}(x,y) = d_3(x,y) \cdot (1 - K_e \Delta t). \tag{16}$$

In nature, the evaporation has a negligible effect, but in our model it is important because the scene would fill up with

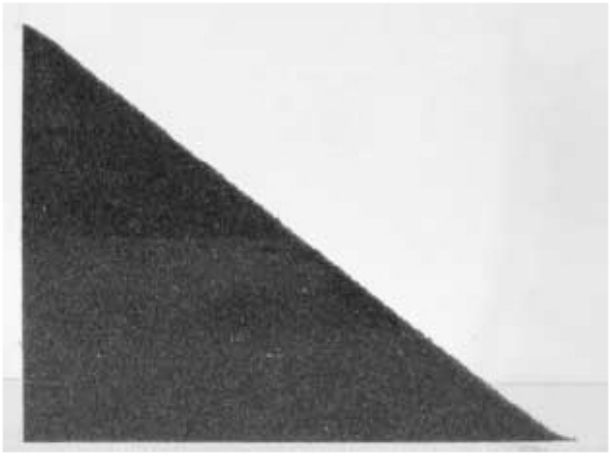


Figure 3: Demonstration of talus angle: photograph of real red sand dumped between flat plexi plates. The material has near linear slope. [13].

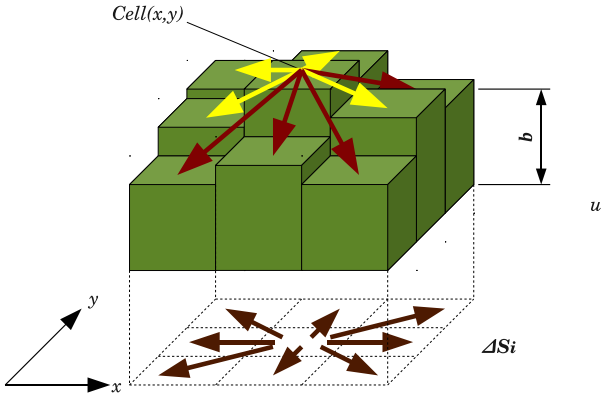


Figure 4: Virtual pipes of thermal erosion. Brown arrows: virtual pipes. Red arrows: soil movement from the $Cell(x,y)$ to the neighbors whose are lower than the talus angle. Yellow arrows: soil movement is inhibited to cells which height is above the talus angle.

water by the time if the water was not removed from the system.

Our thermal erosion model is based on [9]. The original method was designed neither to run on parallel architecture nor in real-time, but with some improvements these problems can be solved. Let's denote the terrain height of the current (x,y) cell by b and its eight neighbors by $b^i, i = 1,2,...,8$. Let's denote the height difference between the current cell and its lowest neighbor by $H = \max\{b - b^i, i = 1,...,8\}$. The area of each of the cells is a and the volume to be moved is $\Delta S = a \cdot H/2$. This is the maximum, otherwise the algorithm will oscillate. To handle local $R(x,y)$ terrain hardness in cell (x,y), we have extended this formula:

$$\Delta S_{t+\Delta t} = a \cdot \Delta t \cdot K_t \cdot R_t(x,y) \cdot H/2 \qquad (17)$$

where K_t is a global coefficient. Then we move this amount to the lower neighbors proportionally if the so called talus angle is larger than that the value determined by material viscosity. The talus angle is an important static parameter of solid granular materials without cohesion between grain particles. To measure this parameter, we should dump the material slowly to a flat surface between two transparent plates. The material will form a slope with an angle, which is a maximum that the given material can reach. Above this angle, the material starts moving to lower levels, making the slope lower. When slope angle reaches this critical angle, then the material does not move anymore [13]. See figure 3.

Let's denote the distance between two cells by d and talus angle by $\alpha = tan((b - b^i)/d)$. Let's denote the set of neighbors that are lying lower than the current element under the talus angle by $A = \{b^i, b - b^i < 0 \wedge tan(\alpha) > (R(x,y) * K_a + K_i), i = 1,...8\}$, where K_a and K_i are global simulation parameters controlling minimum talus angle dependence on $R(x,y)$ local hardness factor. Each element in A will get part of the volume ΔS_i proportional to its height difference:

$$\Delta S_i = \Delta S \frac{b^i}{\sum_{\forall b^k \in A} b^k}. \qquad (18)$$

In contrary to the original model, we do not move ΔS_i volumes directly to cells in set A because this would introduce data write dependency that we wanted to avoid for easy parallel execution. Similarly to fluid flow simulation, we put these quantities into eight virtual pipes carrying material to the neighbors of this cell (see Figure 4), then a separate simulation step updates the terrain height for each cell by summarizing the incoming material flow from their neighbors.

With this modification we can easily execute the algorithm in parallel and integrate it to the fluid-based erosion simulation:

1. Water incrementation due to rain or water sources.
2. Flow simulation using shallow-water model. Computation of velocity field and water height changes.
3. **Soil flow calculation with outflow in virtual pipes of thermal erosion model**.
4. Simulation of erosion-deposition process.
5. Transportation of suspended sediment by the velocity field.
6. **Thermal erosion material amount calculation**.
7. Water evaporation.

With these improvements, we can execute the two erosion models in conjunction with each other. Steep walls of riverbeds carved by hydraulic erosion will start to fall

down due to thermal erosion. And vice versa, the material eroded by thermal erosion will be dissolved and transported by the running water.

3 GPU Implementation and Visualization

In our implementation, the cell structure is represented by 2D 4-channel floating-point texture layers stacked upon each other, attached to a single framebuffer object. Texels of the same position in texture layers form one cell, containing all the simulation variables associated with it. With two such framebuffers we calculate one iteration using one of these buffers as an input and the other as an output (ping-pong) [12]. After the iteration we swap these buffers and start over. The iteration process is implemented in a single fragment shader that runs in three passes on a full-size quad rendered over the entire framebuffer, writing the output textures using the multiple render target feature. The implementation exploits linear interpolation and edge wrapping, which are basic features of the graphics hardware.

The texture buffers can be used directly to visualize the terrain height values to offset triangle vertices of the rendered mesh in the y direction. For the sake of simplicity we used a regular grid mesh to render the terrain surface, but heightfield texture can also be used in advanced terrain rendering methods.

Water is rendered in a similar way. The water surface is a simple uniform grid mesh too, rendered with y offset with the water height and a small $-\Delta y$ constant. Due to z-buffering this makes water surface invisible where water height does not exceed Δy. Water surface has a simple fragment shader that calculates simple sky reflections and alpha transparency.

4 Results

We have used regular and randomly generated terrains to test our method. Regular terrains were utilized to test our parallel erosion methods (see Figure 5 and Figure 6) and see they are running without problems. As we can see, thermal erosion makes the terrain material spread around until the critical talus angle is reached, then the material stops moving. Hydraulic erosion carves the surface creating deep valleys.

Random terrains were utilized to test the model in natural-like scenarios. Random terrains were generated with the the well-known Diamond-square algorithm (see Figure 7a). We implemented it non-recursively to make it possible to generate terrains with similar patterns at different resolutions. We used Gaussian distributed random numbers to generate middle point heights in the algorithm instead of regularly distributed ones, which makes the result more realistic.

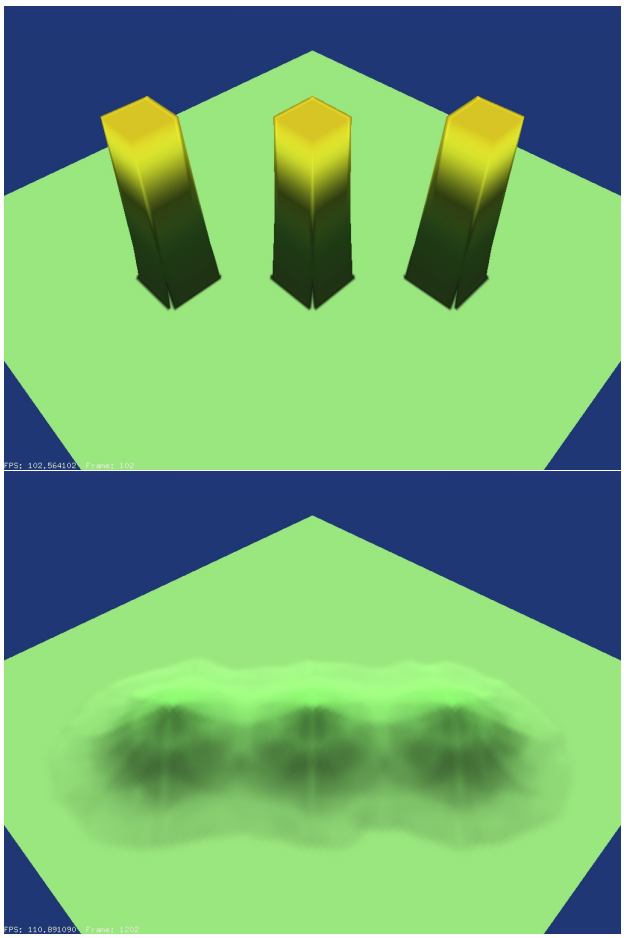


Figure 5: Effects of our parallel thermal erosion method on a regular height field after 1000 iterations.

Our erosion model uses $R(x,y)$ local hardness coefficient in every (x,y) cell. This is a value in range 0..1, that represents the resistance of the soil against in a given cell. Smaller values are representing harder material and vice versa. We generate this coefficient array by copying and scaling the terrain heightfield, adding some random noise to every cell, and applying a global Gaussian filter to the array. The idea behind this was twofold. First, in the nature, the terrain material at higher levels is usually more resistant to erosion than the lower parts. Second, real-world terrain material is never homogeneous, the consistency varies from location to location. With this simple method we can improve the realism of the generated terrain. The surface will be more irregular, similarly to real-world terrains where the material inhomogenity influences the erosion at different locations (see 1).

Figure 7b shows the effects of the improved hydraulic erosion model on the aforementioned random terrain. The hydraulic erosion carves deep grooves into the surface that rarely occurs in nature. This is why we included thermal erosion in the model.

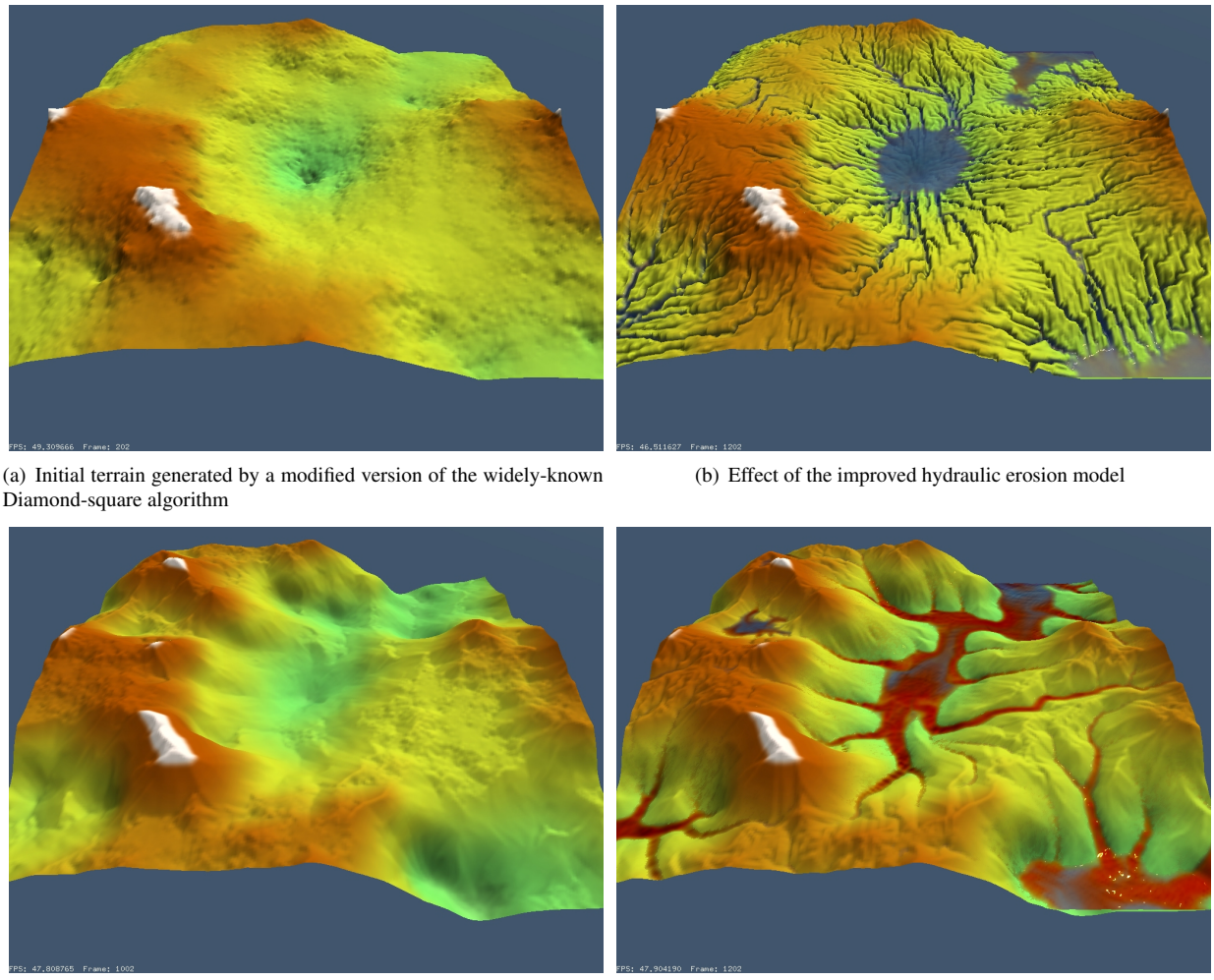
Figure 7c shows the effects of the parallel thermal erosion process on the random terrain. The material at too

Terrain Size	Speed (ms/iteration)				Ratio (GPU/CPU)	
	CPU1	CPU2	GPU1	GPU2	conf. 1	conf. 2
128x128	82.8	25.7	2.57	0.295	32.21	87.1
256x256	325	102.4	9.92	1.01	32.76	101.3
512x512	1289.1	412.6	39.22	3.835	32.86	107.5
1024x1014	5165.6	1647	160.15	15.47	32.54	106.5

Table 1: Performance results

Symbol and Description		Range	Value	Symbol and Description		Range	Value
Δt	Time increment	$[0;0.05]$	0.02	K_s	Soil suspension rate	$[0.1;2]$	0.5
K_r	Rain rate	$[0;0.05]$	0.012	K_d	Sediment deposition rate	$[0.1;3]$	1
K_e	Water evaporation rate	$[0;0.05]$	0.015	K_h	Sediment softening rate	$[0;10]$	5
A	Virtual pipe cross section area	$[0.1;60]$	20	K_{dmax}	Maximal erosion depth	$[0;40]$	10
g	Gravity	$[0.1;20]$	9.81	K_a	Talus angle tangent coeff.	$[0;1]$	0.8
K_c	Sediment capacity	$[0.1;3]$	1	K_i	Talus angle tangent bias	$[0;1]$	0.1
K_t	Thermal erosion rate	$[0;3]$	0.15				

Table 2: Allowed ranges and typical values of global simulation parameters used in our simulator.



(a) Initial terrain generated by a modified version of the widely-known Diamond-square algorithm

(b) Effect of the improved hydraulic erosion model

(c) Effect of our thermal erosion model

(d) Effect of the combined erosion method

Figure 7: Effects of different erosion methods on random terrain after 1000 iterations. The reddish color indicates sediment in the water.

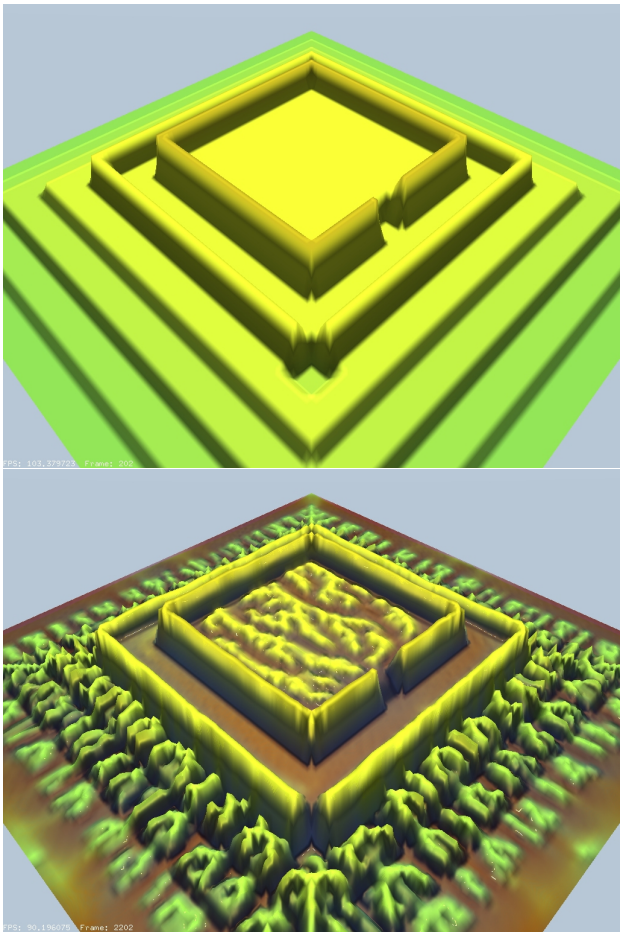


Figure 6: Effects of our hydraulic erosion method on a regular height field after 1000 iterations.

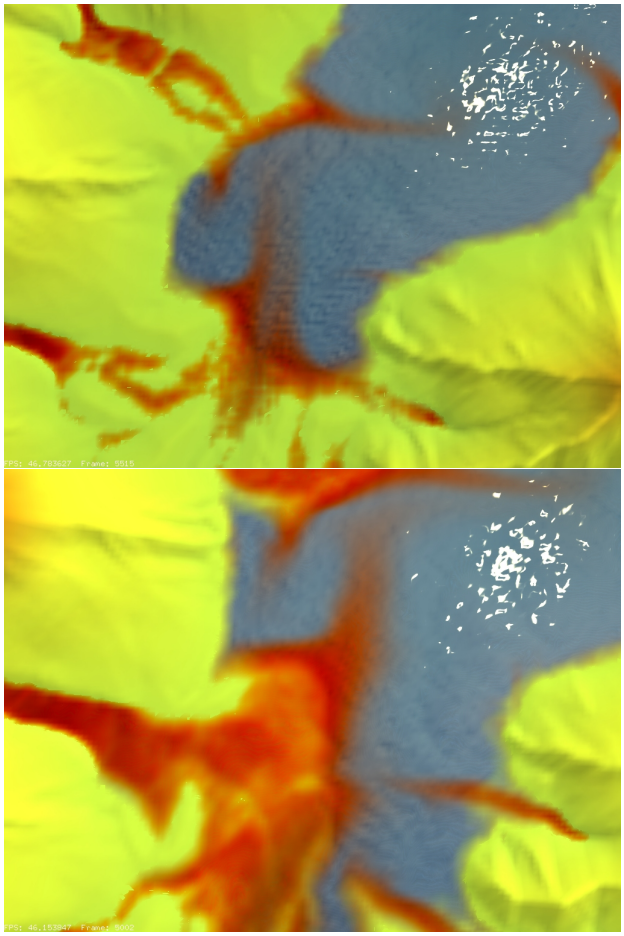


Figure 8: Elimination of the oscillation in the original erosion model.

steep slopes is moving towards the lower regions.

Figure 7d demonstrates the effects of our combined erosion algorithm on a random terrain. We can observe that the riverbeds and valleys are v-shaped which is more realistic than in the original erosion model.

In our framework the following ranges were defined for global simulation parameters:

Figure 8 shows the effect of oscillations on the water surface appearing at long simulations (upper image). These ripples were successfully removed by some modification applied to the model (lower image). See equations 12c and 13c.

Figure 9 illustrates the effect of erosion depth limiting introduced in eqation 10. On the upper image the original model carves riverbeds unrealistically deep. The improved version produces more credible riverbeds (lower picture).

We tested the performance of our model on the two following hardware configurations.

1. CPU1: AMD Athlon XP 3.2 GHz
   GPU1:ATI Radeon HD3650 AGP

2. CPU2: Intel Core2 Q9550 2.83GHz
   GPU2: ATI Radeon HD4870

We implemented the erosion model both on CPU and on GPU to measure the difference between the two architecture. The results are summarized in Table 1. We can see that the speedup gained by utilizing GPU is very high, we reached 30 to 100 times faster execution. On the second configuration, the ratio is higher, showing that the GPUs are developing faster than the ”traditional” serial CPUs due to their more scalable architecture. Using GPU implementation, our algorithm can run at interactive speed even on larger terrains.

5 Conclusions

We proposed an erosion model that can be executed on massively parallel architectures like graphics processors. The method combines and improves two algorithms to simulate hydraulic and thermal erosion in conjunction with each other. The original hydraulic erosion method is extended to be more versatile and stable. The thermal erosion model is a parallel redesign of an earlier work. It makes it able to run on parallel architectures and be integrated with the fluid erosion model. Our method gener-

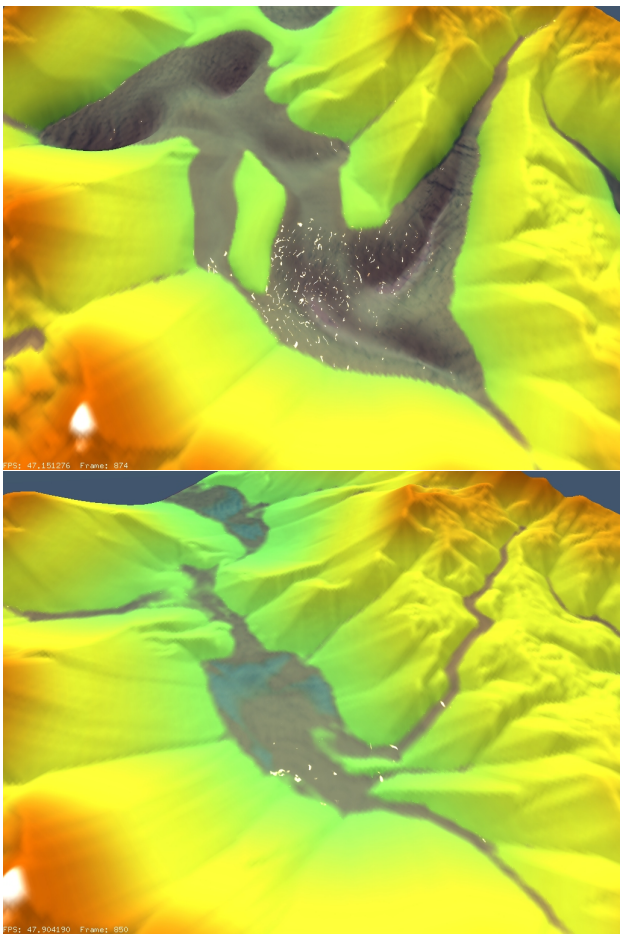


Figure 9: Effects of hydraulic erosion depth limit.

ates more realistic results while still running at interactive speeds.

Acknowledgements

This work has been supported by OTKA K-719922 and by TÁMOP-4.2.1/B-09/1/KMR-2010-0002.

References

[1] Bedřich Beneš. Real-time erosion using shallowwater simulation. *The 4th Workshop on Virtual Reality Interactions and Physical Simulation - Vriphys'07*, pages 43–50, 2007.

[2] Bedřich Beneš and Rafael Forsbach. Layered data representation for visual simulation of terrain erosion. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, page 80, Washington, DC, USA, 2001. IEEE Computer Society.

[3] Bedřich Beneš, Václav Těšínský, Jan Hornyš, and Sanjiv K. Bhatia. Hydraulic erosion: Research articles. *Comput. Animat. Virtual Worlds*, 17(2):99–108, 2006.

[4] Norishige Chiba, Kazunobu Muraoka, and Kunihiko Fujita. An erosion model based on velocity fields for the visual simulation of mountain scenery. *Journal of Visualization and Computer Animation*, 9:185–194, 1998.

[5] E. Galin, P. Poulin (editors, B. Neidhold, M. Wacker, and O. Deussen. Interactive physically based fluid and erosion simulation.

[6] Mark Harris. Fast fluid dynamics simulation on the gpu. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[7] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, volume 24 of *SIGGRAPH '90*, pages 49–57, New York, NY, USA, September 1990. ACM.

[8] Xing Mei, Philippe Decaudin, and Bao-Gang Hu. Fast hydraulic erosion simulation and visualization on GPU. In *15th Pacific Conference on Computer Graphics and Applications, Pacific Graphics 2007, November, 2007*, pages 47–56, Maui, Hawaii, Etats-Unis, November 2007. IEEE.

[9] Forest Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains, 1989.

[10] James F. O'Brien and Jessica Kate Hodgins. Dynamic simulation of splashing fluids. In *Proceedings of the Computer Animation*, pages 198–, Washington, DC, USA, 1995. IEEE Computer Society.

[11] László Szirmay-Kalos and László Szécsi. *General Purpose Computing on Graphics Processing Units*. MondAT kiadó, 2011.

[12] László Szirmay-Kalos, László Szécsi, and Mateu Sbert. *GPU-Based Techniques for Global Illumination Effects*. Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers, 2008.

[13] Péter Vankó. Izgalmas mérések a mérnök-fizikus hallgatói laboratóriumban. *Fizikai szemle, BME TTK*, 9:307, 2006.

[14] Enhua Wu, Youquan Liu, and Xuehui Liu. An improved study of real-time fluid simulation on gpu. *Department of Computer Science, University of Manchester, UK. Since*, 15:139–146, 2004.

Visualization

Maximum Intensity Projection Weighted by Statistical Cues

Peter Mindek*
Supervised by: Ing. Peter Kapec†

Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Bratislava / Slovakia

Abstract

Volumetric visualization of medical data is a specific task, as doctors and radiology technicians are not well trained in the field of computer graphics; therefore, algorithms for visualization of medical data must be as intuitive as possible, so that visualization tools employing them would be helpful in medical environment. Visualization of volumetric data acquired by medical imaging could not be effectively used without defining a proper transfer function, which transforms measured intensities to colours and opacity values. Many visualization methods use complex means for designing transfer functions, which could lead to decreased usability of said methods. We propose a modification of MIP, a common volumetric visualization method that uses a simple one-dimensional transfer function for classifying different materials. The goal of our method is to visualize individual tissues from the medical data, present them with minimal effort and enable users to observe areas of interest.

Keywords: Medical Data, Volumetric Visualization, Classification, Maximum Intensity Projection, Standard Deviation

1 Introduction

Volumetric visualization plays important role in medical imaging. Data acquired by CT (Computed Tomography) or MRI (Magnetic Resonance Imaging) scanners can be visualized using two-dimensional cross-sections and examined by moving these cross-sections through the data set. Another way is to visualize the data set as a whole by means of volumetric visualization. This approach has the advantage of displaying the data in their original context and it is better suited for some tasks.

Direct volume rendering (alpha blending accumulation of sampled values in front-to-back or back-to-front order) is a commonly used method for volumetric visualization. It is usually implemented using the volumteric ray casting algorithm. One of the disadvantages of this method is that it is rather slow (however, it can be accelerated by precalculation of gradients used for lighting) and it needs a proper transfer function, which is essential for the algorithm. It has to be properly designed so that it will correctly classify individual organs, tissues or materials represented by the volumetric data set. Designing such function is not a trivial task and it is often time consuming when done manually; consequently, it is not uncommon to use simpler methods of transforming intensities sampled from volume data along the viewing ray into pixel colour.

One possible solution is to visualize an iso-surface extracted from the visualized data set. [1] Major drawback of this rendering method is that only a small part of the data can be examined at once. Users do not see through the iso-surface, nor do they see anything above the rendered surface; therefore, this method is not suitable for visualizing complicated biological structures or tissues of human body, as they usually do not consist of voxels of the same intensity. However, it can be used to visualize bones in CT scans, or parts of objects scanned by industrial CT scanners.

In medicine, it is often necessary to display structures composed of materials with different intensity values. Where the direct volume rendering or the iso-surface extraction is not an option, simple visualization methods based on processing every sampled value along the rays can be used. A common way is to use some statistical property of the one-dimensional data sampled along the ray. This could be for instance standard deviation, mean value, or maximum value. Using the maximum value is a widespread method called Maximum intensity projection (MIP) [12]. These methods are non-photorealistic, as their goal is not to necessarily mimic real appearance of the visualized objects but rather provide adequate insight into the volumetric data.

Since these methods generally do not use any shading of sampled voxels, their advantage over direct volume rendering is in their speed. The speed enables these methods to be implemented in real time. This is important for usability of the rendering method, as dynamically changing view or other properties of the visualization is crucial for the medical imaging applications.

*mindek06@student.fiit.stuba.sk
†kapec@fiit.stuba.sk

[1] This could be done without ray casting as well; algorithms such as Marching cubes could be employed to extract the iso-surface and generate a polygon mesh that would be rendered using standard rendering capabilities.

Another major advantage of the MIP algorithm and similar methods is their simplicity. Very easy set-up of simple transfer functions or similar classification methods usually results in satisfactory images.

On the other hand, using only the statistical properties of the sampled data or an order independent-operators (such as a maximum operator used in the MIP rendering) has a drawback of discarding depth information of the visualized objects. Many different algorithms have been developed in order to maximize displayed information using these simple rendering methods [4, 11, 10, 1].

Additionally, single statistical property may not provide satisfactory representation of data of interest. For instance, the projection of maximum intensity is useful only when higher intensities are those of interest. This may be the case in the CT scans when the bones are to be visualized (bones have generally higher intensity than surrounding tissue), or in scans with application of contrast agents. However, areas of interest are often occluded by structures of higher intensity values in the MRI scans without contrast; therefore, MIP is not applicable in this case.

Sometimes the volumetric visualization algorithms use classification to enhance the final image [9]. The classification assigns colours, opacity values, or both (depends on the rendering method) to individual tissues or materials to help the user to easily distinguish them. One of the simplest and most intuitive methods of the classification is a one-dimensional transfer function. It is able to filter out some of the structures and reveal the areas of interest within the volumetric data set. However, blurryness and other factors cause this classification method to fail on some of the data sets when using MIP or similar visualization algorithms.

2 Related work

As one of the most commonly used volumetric visualization methods, the MIP rendering algorihtm (introduced in [12]) has been subject to many improvements. Its advantages make this method interesting, even though more complex visualization methods exist. Several enhancements that eliminate various disadvantages of MIP have been proposed.

Use of depth weighting has been proposed in [4]. Intensity values are weighted by a value dependant on a distance from the origin (position of virtual camera). This is essentially a fog effect applied to the MIP rendering. It results in images with distinguishable depth of individual visualized objects. However, in some cases it may partially occlude the areas of interest.

Another modification of the MIP method was proposed in [11]. Instead of using the global maximum of the values along the viewing ray, a first local maximum higher than a pre-selected threshold value is used. If no value is higher than the threshold, the global maximum is projected. An improvement of this method has been proposed in [10].

A method that combines advantages of MIP and direct volume rendering has been introduced in [1]. This method updates an opacity profile based on difference between the sampled value and a current maximum value.

Numerous volumetric visualization methods that provide additional information about visualized data have been introduced as well. The goal of these methods is to show as much information as possible without difficult transfer function design. The use of weighted distance transfrom has been proposed to enhance various features in rendered images of anatomical structures and to provide contextual information about selected body parts [5]. The gradient based rendering technique of objects boundaries was introduced in [2]. Additionally, visualization enhancement by level lines has been proposed in the paper. Probabilistic classification of different materials using several one-dimensional transfer functions has been proposed in [8].

Even though the classification methods using the one-dimensional transfer functions are widely used, volumetric rendering methods with multidimensional transfer functions are being examined as well. The problem of designing the multidimensional transfer functions has been also addressed [7]. Semi-automatic generation of transfer functions for direct volume rendering of boundaries between different materials within data set has been proposed in [6].

Statistical transfer function space has been introduced in [3] and addresses the problem of overlapping of intensity distributions of different materials that are to be classified. The method extracts statistical properties from the data and uses them to classify different materials. Described method uses adaptive growing approach to estimate statistical properties of each sample point. Estimation is based on neighbouring values. Extracted properties are also utilized to improve visual quality of volume shading by noise reduction.

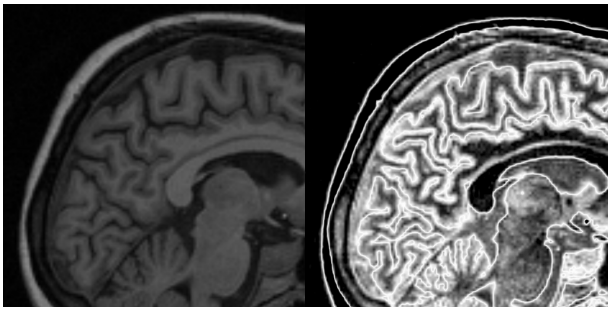
3 Weighting by statistical cues

Our proposed approach is an enhancement of the standard MIP rendering with the distance weighting using the volumetric ray casting algorithm. The goal was to enhance the MIP rendering technique in such manner that it could be employed in rendering of CT or MRI scans with different areas of interest.

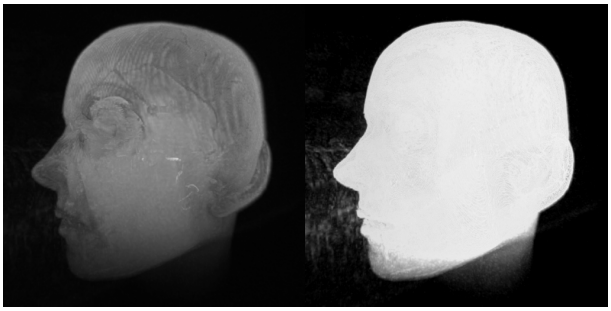
The algorithm uses a one-dimensional transfer function for material classification, which is designed by the user by placing and moving control points of a cubic spline. The transfer function assigns opacity values to individual voxels according to their intensity, which is used as an input for the transfer function. It could be used to suppress various ranges of intensities in the data set to reveal areas of interest.

Figure 1 shows the inability of a one-dimensional transfer function to classify a MRI scan in order to reveal the

brain in the MIP rendered image. The goal was to remove the skull (assign it a zero opacity), which the one-dimensional transfer function failed to accomplish, as the voxels forming the outlines of the skull have the same intensity values as the brain tissue. Thin layers of voxels at the surface of the head after classification have the same or higher intensities than the brain tissue. MIP projects these voxels instead of the area of interest, which is the brain in this case.



(a) Cross-section without classifica-(b) Cross-section with classification tion



(c) MIP without classification (d) MIP with classification

Figure 1: Cross-section through an MRI scan of a human head without classification (1a) and with classification by a 1D transfer function (1b). Brain is occluded in the MIP rendering (1c), as well as in the MIP rendering with classification by the 1D transfer function (1d). In both renderings, higher intensity voxels of the skull occlude the brain.

Our algorithm tries to override this problem by using standard deviation of voxels along the projection ray to calculate the weight of the sampled values. These values are then treated as in the standard MIP rendering – the maximum of the weighted values is projected into the final image.

The standard deviation is calculated from n previously sampled values. These values form a window, whose size is constant during the rendering process. The size of the window should be as big as possible; however, voxel intensities should not be affected by voxels representing different materials. Bigger window size therefore requires use of a smaller step size for the ray traversing.

The algorithm uses parameter τ, which is adjustable along with the transfer function by the user. Weights of the sampled values are calculated by formula 1.

$$w_i = |2\sigma - \tau| \tag{1}$$

σ is the standard deviation of the window of i-th sample point, w_i is the weight. The sampled value is multiplied by the weight and finally, the maximum of the resulting weighted values along the ray is projected into the image.

The τ parameter can be adjusted by the user to further specify the area of interest. It serves as a parameter of a simple V-shaped transfer function for the standard deviation used to weight the sampled intensities from the data set. Appropriate value of the τ parameter can be found by trial and error by continually changing it while observing the final rendering.

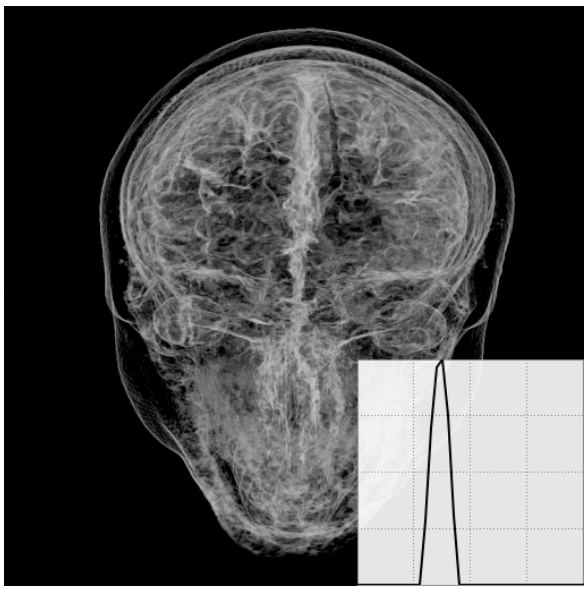
Our algorithm also uses the distance weighting immediately after sampling the value from the data set, as proposed in [4]. The depth of rendered objects is more perceptible this way. Figure 2 demonstrates the ability of our method to efficiently employ a 1D transfer function for material classification.

3.1 Implementation

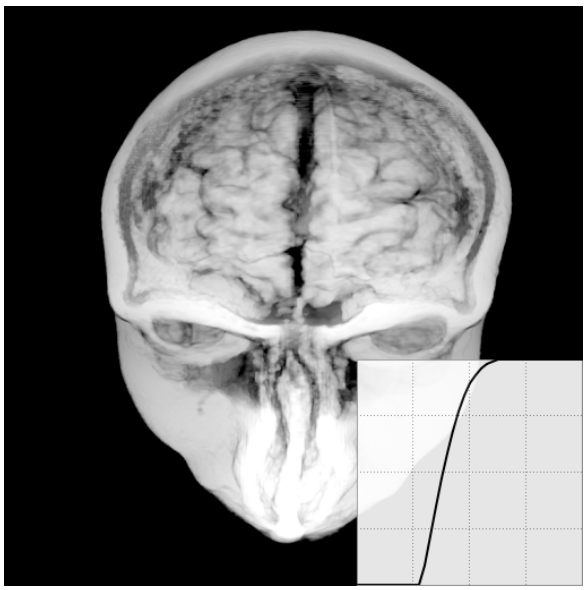
The proposed method is based on the volumetric ray casting algorithm and processes one ray at the time. Colours of individual pixels in rendered image could be calculated independently; therefore, rendering could be parallelized. Implementation on state-of-the-art graphic hardware may take advantage of its massive parallelism and may result in interactive real time rendering.

We have implemented the method as a GLSL vertex/fragment shader pair using OpenGL library. The image is rendered as a single quadrilateral covering the whole screen while the vertex/fragment shader pair is in use. The vertex shader does not transform positions of rendered vertices with the model view matrix, but it uses this transformation to calculate the direction and position of the virtual camera. This design enables the use of the standard OpenGL matrix transformation commands to control the virtual camera of the shader.

The volume ray casting algorithm is implemented as a fragment shader described bellow. The fragment shader processes rays, as fragment colour is dependant only on the evaluation of its respective ray. Number of steps is taken in order to evaluate each ray. Every step consists of sampling two values along the ray. First value is sampled at the sampling position, which gradually move along the ray in constant intervals away from the camera position. Second value is sampled at the position n steps back. The steps, or the sampled values between these two positions are referred to as a window. The two sampled values (which are transformed by the transfer function, stored as a 1D texture) are used to calculate the standard deviation of the window. Standard deviation is calculated by following formula:



(a) Boundaries



(b) Volumes

Figure 2: Comparison of a visualization of object boundaries and volumes by using our method with different transfer functions (x-axis of the transfer function represents sampled intensity, y-axis represents output opacity) ; $\tau = 0$ for both images.

$$\sigma = \sqrt{\frac{n(\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x_i)^2}{n(n-1)}} \qquad (2)$$

$x_1 ... x_n$ are sampled intensities of the current window. The are calculated by trilinear interpolation of voxel values surrounding the sampling point.

To calculate the standard deviation in $O(1)$ time for the window of every sampling point, values of $\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n} x_i^2$ are being accumulated as the shader marches along the ray. In every step, value sampled at the end of the window (farther from the camera) and its square are added to respective accumulators, while value at the begining of the window (closest to the camera) and its square are subtracted from the accumulators. This way, the formula 2 could be used to calculate the standard deviation for every window in constant time, as the accumulators would always contain the sum of the sampled values and the sum of their squares. Consequently, the window would not be centred at the current sampling point.

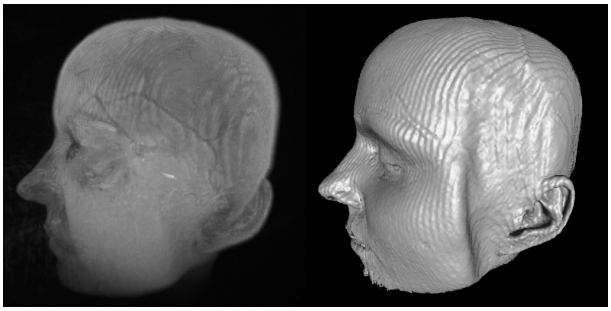
The volumetric data is enclosed in a bounding box in order to speed up the rendering. At the beginning of the evaluation of a ray, two intersections of the ray with the bounding box are calculated (in case the ray hits a vertex of the bounding box, these two intersections are equal). The sampling occurs only between these two positions in space. As the values are sampled from positions with uniform distances between each other, the number of steps for each ray vary. This helps to reduce the rendering time.

The volumetric data are stored as a 3D texture in the memory of the graphic card. This way, sub-voxel values (values from intermediate space between voxels) can be sampled from the data set. Using OpenGL commands, graphic card could be instructed to use the trilinear filtering for the sampling. The trilinear interpolation of values of neighbouring voxels significantly improves the rendering quality, even though it could introduce some minor artefacts to the visualized data. Listing 1 shows a fragment of the shader program implementing out method.

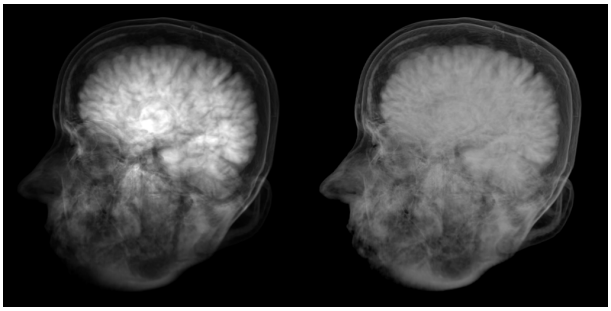
Listing 1: Part of the MIPWSC fragment shader; *p0* and *p1* are positions on the ray, *i0* and *i1* are step numbers on the begining and the end of the window, *t* is the τ parameter.

```
vox0 = dataRead(p0) * (1.0 - float(i0) /
    fogLen);
vox1 = dataRead(p1) * (1.0 - float(i1) /
    fogLen);

proj1 += vox0;
projSqr += vox0 * vox0;
proj1 -= vox1;
projSqr -= vox1 * vox1;
counter = float(i0 - i1);

s = pow((counter * projSqr - proj1 * proj1
    ) / (counter * (counter - 1.0)), 0.5);
vox0 = vox0 * abs(s * 2.0 - t);
if (vox0 > proj) {
    proj = vox0;
}
```

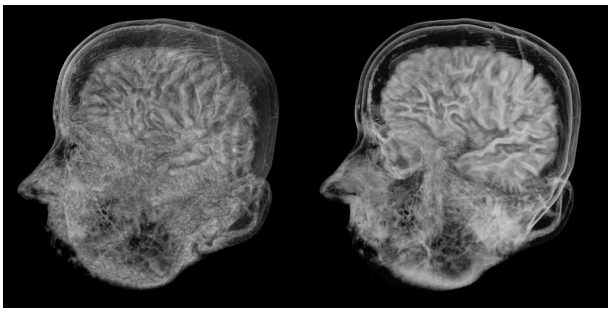
The listed fragment calculates the standard deviation of the window and weights the actual sampled value. The



(a) MIP (b) Iso-surface

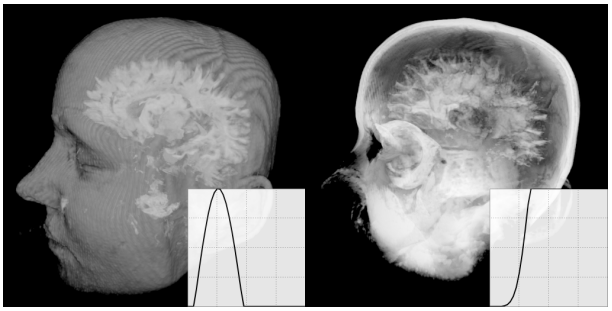


(c) AIP (d) SDP

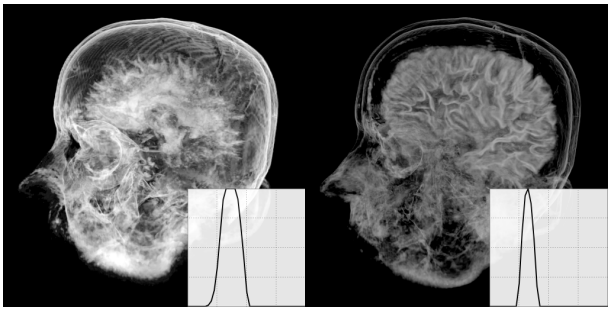


(e) DVR (f) MIPWSC

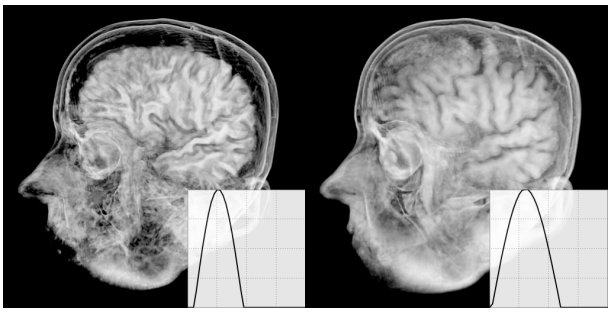
Figure 3: An MRI scan of a female head rendered with different volume visualization techniques using 1D transfer function for classification.



(a) $\tau = 0.6$ (b) $\tau = 0.3$



(c) $\tau = 0.3$ (d) $\tau = 0.2$



(e) $\tau = 0.2$ (f) $\tau = 0$

Figure 4: An MRI scan of a female head rendered using our method with different settings.

conditional branching at the end of the listing serves as a maximum operator for weighted values.

4 Results

As mentioned in Section 3, out method uses a one-dimensional transfer function for transforming intensities of voxels, window with adjustable length, and the τ parameter adjustable by users; consequently, great variety of result images could be achieved from a single data set. The transfer function and the τ parameter can be dynamically modified and are applied on the final rendering in real time. This enables users to find the ideal transfer function and the value of τ by trial and error.

Figure 3 compares several volumetric visualization methods with our proposed method. Drawbacks of individual rendering techniques are demonstrated: maximum intensity projection (3a) and iso-surface rendering (3b) are unable to efficiently use the transfer function classification to reveal the brain tissue. Average intensity projection (3c) and standard deviation projection (3d) can reveal brain occluded by skull, but the final images are blurry and do not show too much detail. Direct volume rendering (3e) is able to show the brain tissue in higher detail using a simple 1D transfer function, but the rendering quality is decreased by severe artefacts. Maximum intensity projection weighted by statistical cues (3f) is able to reveal brain in high detail; therefore, it is demonstrated that our method overrides some of the drawbacks of other commonly used visualization methods.

Figure 4 shows several images rendered using our visualization method as a demonstration of variability, which enables users to classify data in required fashion. One-dimensional transfer function and τ parameter are used to

reveal different tissues.

Figures 5, 6 and 7 show MRI and CT scans of various human body parts rendered using the standard MIP method and our proposed method. One-dimensional transfer functions were used for the classification. The MIP algorithm failed to reveal areas of interest. Our algorithm was able to visualize the structural characteristics of individual objects in the scans.

The algorithm was tested on NVIDIA GeForce GTX 260. We have achieved interactive framerates (40 frames per second at 800x600 screen resolution) on the data sets containing 256^3 and 512^3 voxels. However, rendering speed depends on screen resolution.

5 Conclusions

We proposed a non-photorealistic method for visualization of volumetric data sets. Our approach is an improvement of the MIP rendering method with the goal of being simple, intuitive and fast, yet able to visualize different tissues or materials represented by the volumetric data. We have implemented the method in an interactive prototype application that uses real-time shader programs. We presented a comparison with commonly used methods of volumetric visualization. Results show that our method uses one-dimensional transfer function for classification more effectively and with better results than the other methods.

The algorithm could be improved in several ways. The length of the window used to calculate standard deviation could be adaptively changed during every step according to neighbouring voxels. This would make the algorithm more intuitive, although control over resulting renderings would be probably reduced in some cases.

Acknowledgements

This work was partially supported by the grant KEGA 244-022STU-4/2010: Support for Parallel and Distributed Computing Education. Volumetric data sets were obtained at `http://www9.informatik.uni-erlangen.de/External/vollib/`.

References

[1] Stefan Bruckner and Meister Eduard Gröller. Instant volume visualization using maximum intensity difference accumulation. *Computer Graphics Forum*, 28(3):775–782, June 2009.

[2] Balázs Csebfalvi, Lukas Mroz, Helwig Hauser, Andreas König, and Meister Eduard Gröller. Fast visualization of object contours by non-photorealistic volume rendering. Technical Report TR-186-2-01-09, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, April 2001. human contact: technical-report@cg.tuwien.ac.at.

[3] Martin Haidacher, Daniel Patel, Stefan Bruckner, Armin Kanitsar, and Meister Eduard Gröller. Volume visualization based on statistical transfer-function spaces. In *Proceedings of the IEEE Pacific Visualization 2010*, pages 17–24, March 2010.

[4] Wolfgang Heidrich, Michael McCool, and John Stevens. Interactive maximum projection volume rendering. In *Proceedings of the 6th conference on Visualization '95*, VIS '95, pages 11–18, Washington, DC, USA, 1995. IEEE Computer Society.

[5] Thomas Kerwin, Han-Wei Shen andf Brad Hittle, Don Stredney, and Gregory Wiet. Anatomical volume visualization with weighted distance fields. In *Eurographics Workshop on Visual Computing for Biology and Medicine*, 2010.

[6] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, VVS '98, pages 79–86, New York, NY, USA, 1998. ACM.

[7] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8:270–285, July 2002.

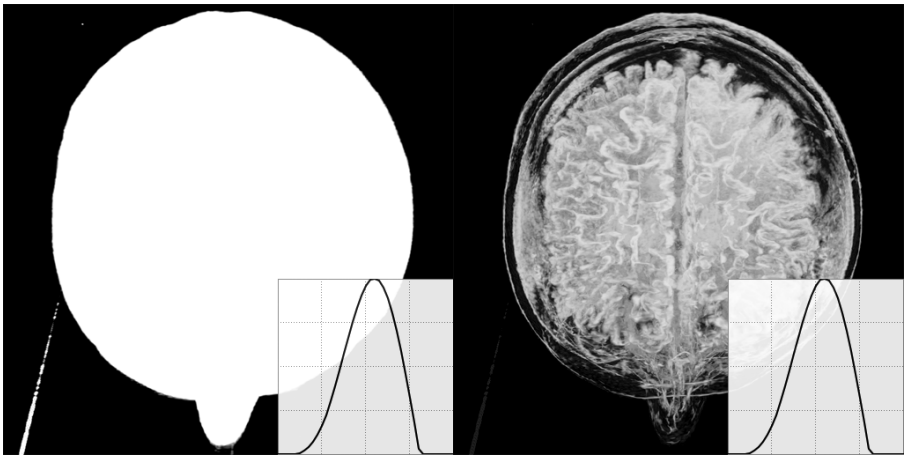
[8] Joe M. Kniss, Robert Van Uitert, Abraham Stephens, Guo-Shi Li, Tolga Tasdizen, and Charles Hansen. Statistically quantitative volume visualization. In *Proceedings of IEEE Visualization 2005*, pages 287–294, 2005.

[9] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8:29–37, May 1988.

[10] Feng Ling and Ling Yang. Improved on maximum intensity projection. *Artificial Intelligence and Computational Intelligence, International Conference on*, 4:491–495, 2009.

[11] Yoshinobu Sato, Yoshinobu Sato Phd, Shin Nakajima, Nobuyuki Shiraga, Shinichi Tamura Phd, and Ron Kikinis Md. Local maximum intensity projection (lmip): A new rendering method for vascular visualization. *Journal of computer assisted tomography*, 22(6):912–917, 1998.

[12] Jerold W. Wallis, Tom R. Miller, Charles A. Lerner, and Eric C. Kleerup. Three-dimensional display in nuclear medicine. *IEEE Transactions on Medical Imaging*, 8(4):297–303, 1989.

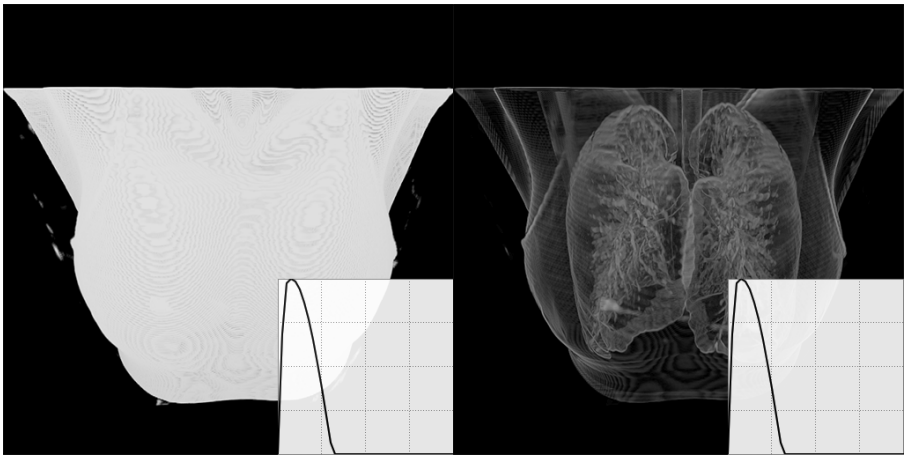


(a) MIP (b) MIPWSC, $\tau = 0.6$

Figure 5: A comparison of MIP and MIP weighted by statistical cues rendering of an MRI scan of a male head.

(a) MIP (b) MIPWSC, $\tau = 0.25$

Figure 6: A comparison of MIP and MIP weighted by statistical cues rendering of a CT scan of a knee.



(a) MIP (b) MIPWSC, $\tau = 0.34$

Figure 7: A comparison of MIP and MIP weighted by statistical cues rendering of a CT scan of a chest.

Visualizing the Effects of Logically Combined Filters

Thomas Geymayer*

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

Abstract

Filtering data is an essential process in a drill-down analysis of large data sets. Filtering can be necessary for several reasons. The main objective for filters is to uncover the relevant subsets of a dataset. Another, equally relevant goal is to reduce a dataset to dimensions to which either visualization or algorithmic analysis techniques scale. However, with multiple filters applied and possibly even logically combined, it becomes difficult for users to judge the effects of a filter chain. In this paper we present a simple, yet effective way to interactively visualize a sequence of filters and logical combinations of these. Such a visualized *filter-pipeline* allows analysts to easily judge the effect of every single filter and also their combination on the data set under investigation and therefore, leads to a faster and more efficient workflow.

We also present an implementation of the proposed technique in an information visualization framework for the life sciences. The technique, however, could be employed in many other information visualization contexts as well.

Keywords: filter-pipeline, brushing, logical operations, interactive, data analysis, compound filter

1 Introduction

Visualizing large amounts of data has been one of the grand challenges of information visualization for over a decade now. With ever more data being produced, the ability to efficiently extract knowledge out of data becomes more important. There are several ways to analyze large quantities of data. Examples are aggregation or drill-down techniques, focus and context methods, and so on. In the sense of visual analytics [12, 7], visualization is combined with computational methods, such as machine learning or statistics. However, in many cases, raw data has several undesired attributes: parts of it can be redundant, noisy or irrelevant for a given task. Also, most methods – either computational or visual – do not scale arbitrarily. Fortunately, there is a simple and yet effective method to reduce the data to a manageable size: filtering. Filtering allows parts of the data to be removed, based on a given criteria. A filter can be defined visually or textually as a processing rule. Filters can be based on fairly simple concepts, such as thresholds, or on more complex processes, such as a statistical evaluation of significance. Related concepts are dynamic querying [1] (selecting only a desired subset of a data set instead of removing undesired parts) and to some extent, also brushing (highlighting a subset of a data set).

It is common to use a combination of filters to continually refine the analysis result. In many cases, such combinations are equivalent to logical operations [10, 5]. While a logical AND is the most commonly used, other operations such as OR, XOR and NOT are feasible as well.

While the reduced data set itself becomes more manageable, the overall filtering process and the individual effects of filters on the data set becomes increasingly obfuscated. To alleviate this, methods to visualize the combination of applied filters have been developed. Hong Chen [4] for example visualizes filters and other parameters of the visualization pipeline in a graph. However, to our knowledge, there has not been any technique that conveys not only the sequence of filters or brushes, but also the effects on the data size. Inspired by Minard's work, the famous *Carte Figurative des pertes successives en hommes de l'arme franaise dans la campagne de Russie 1812-1813* [13], which shows the continuous reduction of men in Napoleon's army during his Russian campaign, we have developed a visualization technique showing the effects of individual filters on a data set.

Our primary contribution is an interactive visualization technique for the effects of multiple filters, including the effects of logical operators applied to combinations of filters. This visualization technique enables users to understand the effects of individual and combined filters. A secondary contribution is a general and detailed analysis of requirements for visualizing multiple filters. Having these requirements at hand, we demonstrate how the proposed technique satisfies each of the specified requirements.

2 Related Work

Much of how we interact with large quantities of data in visualization has its roots in the 1980s and early 1990s. Becker proposed some basic principles for dynamic data analysis [2], like linking & brushing – a technique that is commonly used until today. In 1992 Ahlberg *et al.* [1]

*tomgey@gmail.com

conducted an experiment with dynamic queries performed on a database with different combinations of graphical and textual input and output, respectively. As the different parameters used for the dynamic queries have the effect of refining the data set, this work is an early example of dynamically adapted filters.

In 1995 Martin and Ward propose an improvement of the XmdvTool which contains methods to combine multiple brush operations with different logical operations [10].

A recent example of an approach of filtering high dimensional data can be found in [14] where cross-filtering across multiple views is presented.

These works lay the foundation for modern visualization systems which widely employ combined filters or brushes to refine data sets or selections. However, only very few systems visualize these combinations in an explicit way. One notable exception is the work by Hong Chen [4] where node-link diagrams are used to visualize operations like a brush or selections. He also employs combined nodes which visualize logical or analytical operations. However, while individual operations are visualized, the effects of the operations on the data are not.

3 Problem Analysis

Users often find it hard to remember the steps conducted to get a specific result [14]. To support analysts and reduce the required cognitive load, we believe that an explicit representation of the filter sequence helps to understand the interdependencies between and consequences of filters. We elicited the following main requirements for such a filter-pipeline meta-visualization technique:

1. **Show Sequence**
   As filters are typically applied sequentially, it is essential to show the filters in the sequence they were applied.

2. **Show Consequences**
   To allow a user to judge how much data is removed by a filter, a filter visualization needs to show how much elements a filter reduces.

3. **Show and Create Compositions**
   A simple sequence of filters is equal to logical AND operations (*i.e.*, show all elements which are not removed by filter X and filter Y). Other logical operations such as OR and XOR cannot be visualized as easily. It is essential, that such compositions are adequately represented in a dedicated filter visualization technique.
   Additionally to the visualization of composed filters, it should also be possible to create composed filters based on pre-existing filters.

Aside from these main requirements for a filter visualization technique, there are several other requirements which provide added benefit to users:

4. **Modify Filters**
   An interactive filter visualization technique should enable a user to modify a filter (*i.e.*, change its parameters), to remove a filter and to move a filter to another position in the sequence of filters.

5. **Hide Filters**
   In some cases it may be desirable to hide filters. A common example is an initial filter that removes noisy data. If such a filter removes a lot of data items, the consequences of subsequently applied, fine-grained filters become hard to perceive, due to the small change relative to the initial filter. One solution would be to use logarithmic scales for the amount of data removed. However, as log scales are not intuitive, we believe that the ability to hide filters is superior.

6. **Show Filter Efficiency**
   When a filter is visualized in relation to a previous filter, it is impossible to judge its effect on the global data set, since only the effects on the already filtered data set is shown. To make a user understand the consequences and the efficiency of a filter better, an effective filter visualization technique should also enable a user to see how much data a filter would remove from the complete data set.

In the following chapter, we will describe how we address each of these requirements and challenges to create a simple and yet effective filter visualization technique.

4 Method

Similar to the visualization of the reduction of men in Napoleon's army [13], we render each filter as a quadrangle where the left side represents the input and the right side the output of the filter (see Figure 1). The size is chosen in way that the largest visible filter always fills the available height, and the length is equally distributed over all visible filters. The height of the left edge of the filter encodes the elements going in, while the height of the right edge encodes the elements going out of a filter. Consequently, the difference in height (which is known to be the most powerful visual variable [3]) as well as the slope of the top edge allow to easily judge the effect of the filter. To convey a sense of absolute numbers, we also chose to show the number of current elements and the number of elements each filter removes from its input.

4.1 Basic Sequence of Filters

We show a sequence of filters as an equivalent to the logical AND operation, which simply concatenates one filter after each other, such that the output of a filter is passed to the next filter as input. This simple, yet effective method satisfies Requirements 1 and 2.

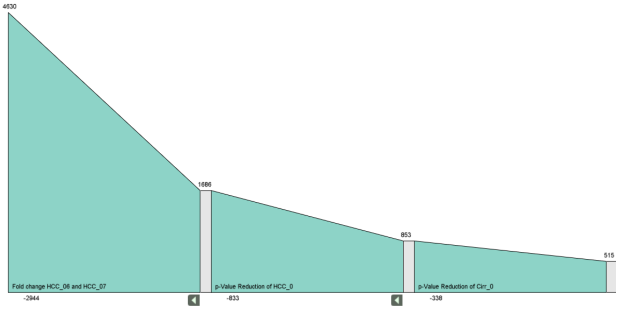


Figure 1: Sequence of filters. The result is equivalent to a logical AND operation of the filters.

4.2 Compound Filters

For more advanced filter-pipelines, combinations of multiple filters into a single filter can be necessary. We provide the possibility to create meta-filters where all involved (sub)-filters are combined in one filter in the sequence of top-level filters. This can be achieved by dragging an existing filter and dropping it onto another already created filter. The whole meta-filter's input data is passed to each sub-filter. The output is then calculated based on the desired logical operations – most commonly an OR.

In order to visualize this combination, we experimented with two different approaches. One is to stack every involved sub-filter on top of each other and embed this stacking into the overall meta-filter. Alternatively, we embed all sub-filters in the top-level meta filter without overlaps. In the following, we will briefly discuss the advantages and disadvantages of each method.

Stacked Sub-Filters

As each sub-filter of the combined filter receives the same input, it is intuitive to render all sub-filters on top of each other at the same location (see Figure 2). The filters are sorted from top to bottom, where the topmost sub-filter (in our example rendered in a light purple color) is the least effective one (the one that removes the least elements from the input data), thus guaranteeing that no filter is completely occluded. The height on the left side of each filter is the total input of the compound filter and the height on the right side of each filter represents its individual output. Additionally, in the background, the union of all filters is rendered, visualizing the total influence of the combined filters on the filter-pipeline. In our example, this is the largest filter with the light yellow background.

It is also of interest, to know which part of the input that passes all filters, which is the intersection of all individual sub-filter outputs (the result of an AND operation). We visualize this by adding another quadrangle on top. According to the characteristics of set intersection this will always be the smallest quadrangle (in Figure 2 it is rendered in green). This information can also be confusing to the user and misinterpreted as an additional filter. Thus, we only show it if the user moves the mouse over the filter. This allows him to detect inefficient filters, *i.e.*, filters that only contribute few or even no elements, apart from the elements also contributed by the other filters.

A problem with this approach is that it becomes cluttered easily. We found that for as little as three filters, especially if they are similar in terms of their efficiency, it is not easy to distinguish individual filters. Furthermore, it discontinues the flow of the data through the filter-pipeline, as all sub-filters have dead ends on their right sides without an equivalent at the left side of the next filter. Consequently, we devised an alternative method which addresses both issues.

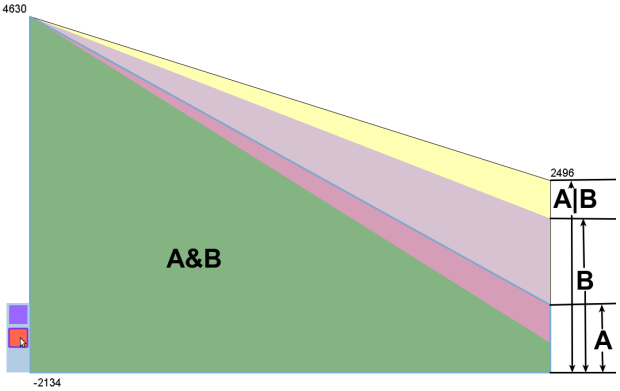


Figure 2: Two filters combined with logical OR, both sub-filters stacked over each other. The largest filter with the light yellow background is the resulting filter (A|B), the two purple filters are the combined sub-filters (just A or just B) and the green filter on top of all filters represents the intersection of the elements filtered by both sub-filters (A&B).

Separate Sub-Filters

As the sub-filters in a compound filter operate in parallel (contrary to the sequence of filters on the top-level), we considered to also express this property in the visualization technique. Consequently, we render the sub-filters at a smaller scale in parallel inside the resulting compound filter. To provide a continuous flow of the data through the filter-pipeline, we connect the input of the compound filter to each sub-filter using curved shapes (see Figure 3). The surfaces use the same color as the respective sub-filters, with transparency increased to allow a user to see the overlapping regions. Inspired by Kosara *et al.*'s work on categorical data visualization [8], we then calculate all possible intersections between the contributed elements of every sub-filter. Consequently, for a composition of two filters, if the underlying operation is an OR, there are two categories of elements. Those which are contributed by only one sub-filter, and those that are contributed by both. We render each set with a trapezoid using the same color as we did for the incoming surfaces (see Figure 3). To make the relative size of the set intersections more obvious, we use the space right of the filter, to show the set sizes in de-

tail. Moving the mouse over an intersection, shows which filters are intersected for this sub-set.

With the technique of using separate sub-filters embedded in a meta-filter, we have successfully addressed Requirement 3.

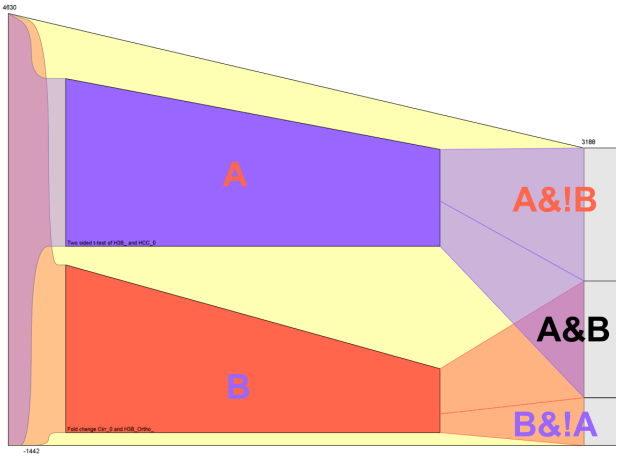


Figure 3: Two filters (labeled with A and B) combined with logical OR, visualized in parallel. The large filter in the background, rendered in light yellow, is the resulting filter. The left edge of the resulting filter is connected with the sub-filters using curved surface. On the right side, all intersections between the elements that are passed to at least one of the sub-filters are visualized – elements contributed only by filter A (A&!B), elements contributed only by filter B (B&!A), and elements contributed by both, filter A and B (A&B).

4.3 Modifying Filters

The described filter visualization lends itself to allow interaction with the filters themselves. As discussed in Requirement 4, the essential operations are: modify, remove and move. We provide intuitive access to those features, for example by drag and drop for moving filters, or by double clicking on a filter for modifying it.

4.4 Hiding Filters

We have already discussed the issue of combinations of strong filters that initially remove large portions of the data, and more sensitive, refining filters that remove only smaller parts (see Section 3, Requirement 5). Another issue, aside from the inability to perceive the effects of filters removing only a view elements, is the fact, that composed meta-filters containing several sub-filters are hard to see because of the tiny amount of space available. These problems are illustrated in Figure 4. As a solution to this problem, we provide the possibility to hide a number of filters at the front of the filter-pipeline. This way, the remaining filters can be scaled up to the whole height, which makes their subtle effects on the filter-pipeline, as well as the embedded filters visible again. The effect is shown in Figure 4.

Below each filter, there is a button that enables the analyst to hide all filters from the front up to the according filter. If at least one filter is hidden, we show a button on the left margin to again display the hidden filters.

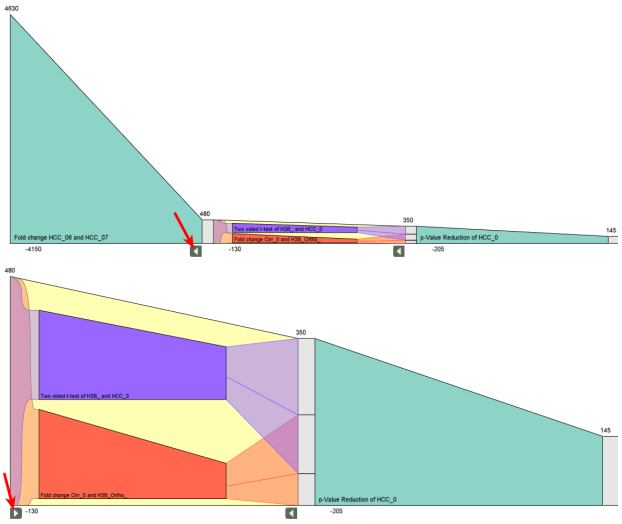


Figure 4: Hiding filters: By pressing the arrow button below the filter, the filter and all filters left of it are hidden. The example shown reduces the visible pipeline to only two filters which are scaled to the whole available height, as depicted in the lower image. Notice that the relative changes and the composition of the compound filter are much better visible when compared to the upper figure. By clicking on the button on the left border, the hidden filter can be made visible again.

4.5 Show Filter Efficiency

As every filter in the pipeline gets the output of its preceding filter, the amount of elements filtered is smaller (in most cases) than if it were applied on the whole input data set. However, as discussed in Section 3, Requirement 6, it can be useful to get an idea on how the filter would behave if it were applied to the whole data set. This, for example, is desirable when the data is filtered to meet a pre-condition for a feasible runtime of a given algorithm. In such a case, a user can apply different filters at the same time, and judge, whether he could meet the requirements with for example only one of the filters.

We address this challenge by overlaying an transparent version of the filter, showing its size as if it was the only filter in the pipeline, on mouse over. This is shown in Figure 5.

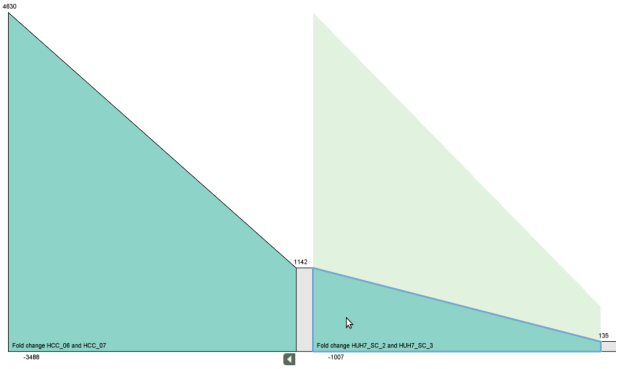


Figure 5: When hovering over the filter, its full size is shown in the background.

Having fulfilled all of the elicited requirements, we will now discuss how the described visualization technique is embedded in an information visualization framework, and give some examples on how the filter-pipeline is used.

5 Visualization Environment

The filter-pipeline view is a part of the Caleydo visualization framework[1] [9], developed at our institute. It is intended to be used for the analysis of large data sets from the life science domain, more specifically genetic and clinical data. Its multiple coordinated view system provides different ways to explore the loaded data set. For example, to explore gene expression data parallel coordinates, a hierarchical heat map, scatterplots and many further views are available. Figure 6 shows a screenshot of a typical analysis session in Caleydo.

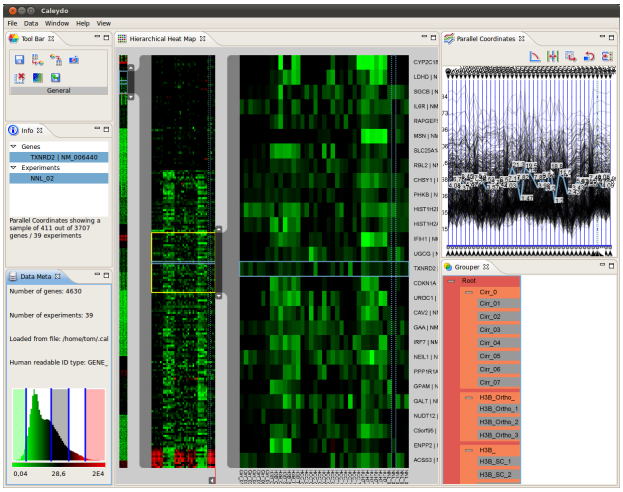


Figure 6: Example of a possible usage of Caleydo with a heat map, parallel coordinates and a view showing the hierarchical grouping of experiments.

As the initially loaded data sets in this domain are often very large, different types of filters are usually applied to reduce the size of the viewed data set, which is especially relevant to enable algorithmic methods. The different views support various ways of brushing and consequently filtering data. In the parallel coordinates, it is possible to filter data where the gene expression values never leave a given interval and therefore, for example, filter genes that are neither over- nor under-expressed. Another possibility to create a filter is to use the angular brush [6] which removes experiments with a deviation exceeding a visually specified threshold from the gene expression value of the selected gene of a specific experiment.

Caleydo also provides computational filters commonly used in gene expression analysis. One example is the fold-change filter that removes all elements which change less than a specified n-fold change to a reference experiment. Other examples are statistical variance tests, which ensure that outliers within control groups, which may be the result of errors in measurement, are removed.

The described filter-pipeline is used in Caleydo to convey the effects of complex combinations of filters. A typical scenario is shown in Figure 7.

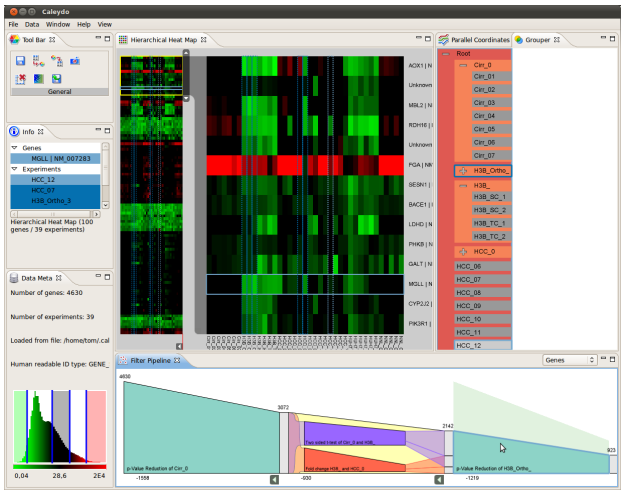


Figure 7: Example of using Caleydo with the described filter-pipeline view opened in the bottom right part of the window.

6 Use Case

In this section, we describe how Caleydo is used for the analysis of gene expression data, acquired in an experiment to find the genetic cause for liver cirrhosis. Liver cirrhosis is known to have a significant genetic component, since, for example only a portion of heavy alcohol abusers actually suffer from it, and conversely, many others, especially those with diabetes suffer from it as well. Our partners at the Medical University of Graz have found, that a specific genotype of mice (*i.e.*, a genetic variant), do not suffer from steatohepatitis, a precursory symptom to liver cirrhosis, when fed with poison over a course of eight weeks, while other genotypes do. They therefore conducted a controlled experiment with the different mouse

[1] http://www.caleydo.org

genotypes, and analyzed their genetic expression after 0 days, 7 days and 8 weeks.

For the analysis they used the Caleydo software. The first step in the analysis is to filter those genes with too much deviation within the control group, thereby ensuring statistical significance of the experiments. The second step is to filter out all genes which are not at least twice as low or twice as high in the 7 days and 8 weeks scenarios, compared to the 0 day values (by using the fold-change filter). The filter-pipeline in this case revealed that only a small portion of genes was actually dropping more than two-fold, the majority of the data was removed by the filter. A a significant part however had a two-fold increase. The biologists then investigated the remaining subset in greater detail by interactively adding, removing and refining filters.

7 Implementation Details

The Caleydo visualization framework [9] is written in Java and is based on the Eclipse Rich Client Platform (RCP)[2]. The framework is designed in a modular manner where a minimal core contains integral parts, such as the data management, the event system, *etc.*. Everything else is outsourced into separate and completely independent plug-ins which communicate with each other by using the core's message-based event mechanism.

Each view can either use the Standard Widget Toolkit (SWT)[3] to create a user-interface by using the default widgets provided by the operating system, or for graphically more advanced or three dimensional user-interfaces use the OpenGL API provided by the Java Bindings for OpenGL (JOGL)[4].

In order to synchronize all views, the data set containing the data to by analyzed is stored centrally, so that each view can access it. View changes are handled first by the view under interaction itself and then propagated to all other views which in turn update their visualization based on the new context.

The statistical filters use the R statistics toolkit [11] for calculating the filter elements which are added to the corresponding list.

8 Conclusions and Future Work

As the amount of data to be analyzed is constantly growing, filtering it is a crucial part in the processing pipeline. Therefore, it is important that an analyst is supported in understanding complex sequences as well as compositions of filters. Visualization of those filters in the proposed filter-pipeline is an ideal tool for this task. It allows to understand even complex combinations of filters, and can be modified interactively until the desired result is achieved. Visualizing a sequence of AND combined filter is straight forward, but complex combinations, like a logical OR operation applied to several filters, require much care in visualization design.

We have presented two ways of visualizing compositions of logical filters (see Figure 8 for a complex scenario with four filters combined in an OR operation). The first one, a simple, stacked rendering of filters has shown to be very cluttered and hard to understand. Consequently we developed an alternative that shows each filter in parallel contained in a compound meta-filter, thereby providing an intuitive representation of a compound filter.

Aside from these main objectives, we have elicited several minor requirements improving the interaction with such filter visualization techniques, and proposed solutions for each of the discussed points.

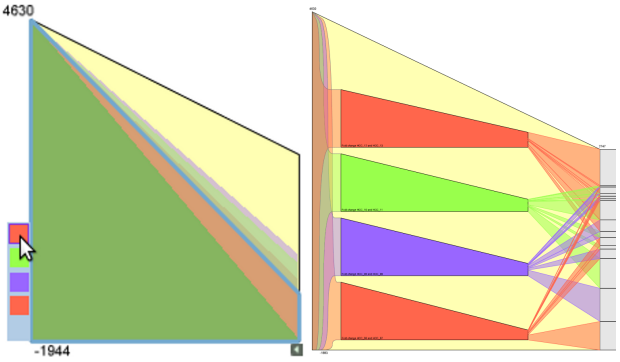


Figure 8: Comparison of different visualizations of compound filters as described in Section 4.2 using four sub-filters.

The support of the complete set of logical operations as well as nested filters, for example by using zoom levels, are promising directions for future research.

9 Acknowledgments

We want to thank our partners from the Medical University of Graz, especially Dr. Karl Kashofer and Prof. Kurt Zatloukal, for providing both data and continuous feedback.

This work was funded in part by the Austrian Research Promotion Agency (FFG) through the *InGenious* project (385567).

References

[1] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. Dynamic queries for information exploration: an implementation and evaluation. *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '92)*, pages 619 – 626, 1992.

[2] http://www.eclipse.org/home/categories/rcp.php
[3] http://www.eclipse.org/swt/
[4] http://jogamp.org/jogl/www/

[2] Richard A. Becker. Dynamic graphics for data analysis. *Statistical Science*, 2(4):355–383, November 1987.

[3] Jacques Bertin. *Semiology of graphics*. University of Wisconsin Press, 1983.

[4] Hong Chen. Compound brushing. In *IEEE Symposium on Information Visualization (InfoVis '03)*, pages 181–188. IEEE Computer Society, 2003.

[5] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the Symposium on Data visualisation 2003*, pages 239–248. Eurographics Association, 2003.

[6] Helwig Hauser, Florian Ledermann, and Helmut Doleisch. Angular brushing of extended parallel coordinates. In *Proceedings on Information Visualization (InfoVis '02)*, pages 127–130. IEEE Computer Society, 2002.

[7] Daniel A. Keim, Joern Kohlhammer, Geoffrey Ellis, and Florian Mansmann, editors. *Mastering The Information Age - Solving Problems with Visual Analytics*. Eurographics, 2010.

[8] Robert Kosara, Fabian Bendix, and Helwig Hauser. Parallel sets: Interactive exploration and visual analysis of categorical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):558–568, 2006.

[9] Alexander Lex, Marc Streit, Ernst Kruijff, and Dieter Schmalstieg. Caleydo: Design and evaluation of a visual analysis framework for gene expression data in its biological context. In *Proceeding of the IEEE Pacific Visualization Symposium (PacificVis '10)*, pages 57–64. IEEE Computer Society, 2010.

[10] Allen R. Martin and Matthew O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings of the Conference on Visualization (Vis '95)*, page 271. IEEE Computer Society, 1995.

[11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. 2010.

[12] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 2005.

[13] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Coneeticut, second edition, 1983.

[14] Chris Weaver. Cross-Filtered views for multidimensional visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):192–204, 2010.

A Problem of Automatic Segmentation of Digital Dental Panoramic X-Ray Images for Forensic Human Identification

Robert Wanat*
Supervised by: Dariusz Frejlichowski†

Faculty of Computer Science and Information Technology
West Pomeranian University of Technology, Szczecin
Szczecin / Poland

Abstract

Dental radiographic images are one of the most popular biometrics used in the process of forensic human identification. This led to the creation of the Automatic Dental Identification System with the goal of decreasing the time it takes to perform a single search in a large database of dental records. A fully automated system identifying people based on dental X-ray images requires a prior segmentation of the radiogram into sections containing a single tooth. In this paper, a novel method for such segmentation is presented, developed for the dental radiographic images depicting the full dentition - pantomograms. The described method utilizes the locations of areas between necks of teeth to determine the separating lines and does not depend on the articulation of gaps between adjacent teeth, thus improving the results achieved in the situation of severe occlusions.

Keywords: image segmentation, dental pantomography, dental human identification, ADIS, forensic identification

1 Introduction

Automatic human identification has always been one of the main applications of pattern recognition. Various biometrics have been used as a basis for such identification, e.g. handwriting, iris, face, fingerprints etc. In reality, there exist situations where some of those biometrics can not be applied. Post-mortem (PM) identification, performed by experts in forensic medicine, consists in determining the identity of a deceased person. This undermines the ability to use some biometrics, such as handwriting or voice, but in some instances there might be other factors rendering other biometrics useless or impractical, for example face recognition in case of fire victims or DNA matching in case of mass natural disasters with multiple casualties. Dental characteristics are popularly used in forensics because of both their robustness to decomposition as well as the speed of a single identification([1]). This led to the creation of the study of proper use of dental evidence in the judicial system - odontology.

The abundance of dental data in criminal cases inspired the Federal Bureau of Investigation (FBI) to create a separate Dental Task Force (DTF) in 1997 ([2]). Its primary task was to create a database to store dental images, known as the Digital Image Repository (DIR), and an automated system for human identification based on the existing Automated Fingerprint Identification System, named Automated Dental Identification System (ADIS). The goal of the system is to provide the ability to narrow the search for an individual in the DIR by automatically finding a small number of the most similar X-ray images. This speeds up the process of a single identification, as out of a large database of images only a small number of comparisons needs to be performed by a forensic expert. The model and functionality of ADIS were presented in [3]. Unlike an identification procedure performed by an odontologist, where artificial dental restorations, such as fillings and dentures, are used as a basis of comparison ([4]), the approach taken in ADIS focuses on teeth morphology and uses teeth contours extracted from dental radiograms in the process of matching.

The simplified model of ADIS assumes three preliminary steps before the comparison: image enhancement, image segmentation and feature extraction ([5]). The first step focuses on improving the contrast of the image, which is usually of low quality. The second step, image segmentation, separates the image into disjunctive segments, each containing at most one tooth. The last step, feature extraction, detects the shape of the contour of a tooth, if present on a given segment, and saves the result in a form that will later be used in the comparison process. While there exist numerous approaches to each of the preliminary steps for intraoral images (i.e. photographs taken with the x-ray film situated inside the patient's mouth, thus showing only a fragment of his dentition), few approaches have been developed for panoramic extraoral images (i.e. photographs taken with both X-ray tube and film moving on an arc on the opposite sides of the patient's head, thus showing the the full dentition). Panoramic images, or pantomograms, convey the largest amount of information of all types of

*robwanat@gmail.com
†dfrejlichowski@wi.zut.edu.pl

Figure 1: A sample pantomogram after image enhancement and cropping.

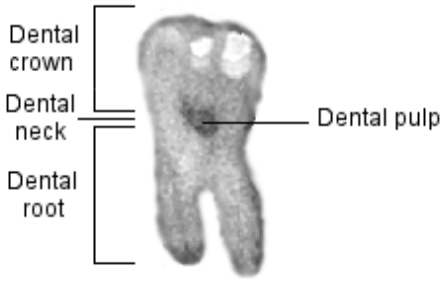


Figure 2: The locations of the parts of a tooth used in this paper.

dental radiographic images because they show the widest range of dentition. The main drawback of this method is that it causes severe occlusions in the resulting image, because it renders a semicircular geometry of the jaw onto a 2-dimensional image. A sample pantomogram that will be used to demonstrate every step of the described method can be seen on fig. 1. All pantomograms in this paper are presented courtesy of Pomeranian University of Medicine in Szczecin.

In this paper, a novel method for image segmentation of a panoramic dental radiogram is proposed. Unlike the previously existing methods for separating intraoral images, the presented method uses two easily localizable points, one between the necks of teeth and one between the roots of teeth (the parts of a tooth mentioned in this paper are shown on fig. 2), in order to determine a straight line delimiting the areas of successive teeth. The method was created to be used with panoramic images only, so it employs features that might be unavailable on intraoral images and should not be utilized with them. Experimental results of the proposed method are also presented on a small sample of radiograms to demonstrate its effectiveness in real world application.

2 Previous work

As has been mentioned in the previous section of this article, there are several existing methods of dental radiogram segmentation. Most of these methods have been created with intraoral images in mind and do not work well with pantomograms.

The first method was presented by Jain and Chen in [6]. It focuses on using integral projections of pixels to detect gaps between teeth. The algorithm is separated into two parts: the first is the detection of the gap between lower and upper jaw, the second is the detection of gaps separating individual teeth. The former step needs user input, so it can be considered semi-automatic. After the initial point of the gap has been selected by the user, moving in both directions, the algorithm chooses short horizontal lines with the highest probability of belonging to the gap. The probability is calculated using (1):

$$p(v_i, D_i) = p_{v_i}(D_i) p_{v_i}(y_i), \tag{1}$$

where $p_{v_i}(D_i)$ is the normalized integral projection of a given horizontal line subtracted from 1 and $p_{v_i}(y_i)$ is a Gaussian with expected value equal to the position of the last chosen line (or the user selection in the first iteration). This probability function has its maximum for the horizontal line that is vertically close to the last selected line and that is composed of pixels with low values. After the maxima have been found for the whole image, a spline function is used to form a smooth line that becomes the separating line between upper and lower jaw. Once the spline has been calculated, for every point on the curve a new integral projection is calculated in the direction perpendicular to its local curvature. These projections assume low values in areas between teeth, thus the search for gaps can be reduced to searching for valleys in the plot of the integrals. The areas between these three lines (gap between upper/lower jaw, two successive gaps between teeth) and the horizontal borders of the image become the segments used later in the process of feature extraction.

The initial tests of this approach revealed that it is not sufficient for separating teeth on pantomograms. The proper operation of this method requires the presence of strongly emphasized dark regions between teeth. This requirement is frequently not upheld in the case of panoramic radiograms. During the research it was determined that in multiple cases integral projections can have lower values in the areas around the middle of a tooth than in the areas on the edge of the tooth, although this situation does not occur on the exemplary radiogram. It happens especially when there are dental fillings on any single side of the tooth or two teeth occlude with each other, creating an area with a heightened radiopacity that increases the integral projection value. Another problem that arises when using the integral projections method is that using the direction perpendicular to the local curvature of the line separating the upper and lower jaw as the slope of a line separating two teeth is only possible with perfectly aligned

teeth, but malaligned teeth can not be separated using this method. While these two shortcomings are not as problematic in the case of intraoral images, their combination resulted in a large amount of mis-segmented images in the database of pantomograms used in this study.

A partial solution to the first problem described in the previous paragraph was presented in [7]. In the case of the lack of strongly emphasized dark regions between teeth the image needs to be processed in order to accentuate the slightly darker regions. 2-dimensional wavelets composed of two 1-dimensional filters were used in [7] to draw out these regions. Based on whether a low-pass filter or high-pass filter was used in each direction, different results are named LL, LH, HL and HH, with the first letter indicating which filter was used horizontally and the second letter indicating which was used vertically. An LH filter is used before the upper and lower jaw separation, while an HL filter is used before the detection of vertical lines separating individual teeth. The segmentation itself is performed using the integral projections.

The last described method, presented in [8], consists of the use of active contour models, also known as snakes. Initially described in [9], snakes are a model of parametrized splines driven towards edges and lines on the image by external forces, i.e. forces derived from the image on which they operate, as well as internal forces, i.e. user imposed control over the elasticity and rigidity of the contour. In [7], a snake used for the segmentation of the image is defined as a parametrized curve $v_s = [x(s), y(s)]$, $s \in [0,1]$ with its energy calculated using (2):

$$E = \int_0^1 (\frac{1}{2}\alpha|v'(s)| + \frac{1}{2}\beta|v''(s)|) + E_{ext}(v(s))ds, \quad (2)$$

where α and β are the weight parameters allowing control of the snake's tension as well as rigidity and E_{ext} is the external driving force along the contour of the snake, given in [8] as (3):

$$E_{ext} = G_\sigma(x,y)I(x,y), \quad (3)$$

where G_σ is a 2-dimensional Gaussian and $I(x,y)$ is the original image. Thus defined external force amounts to the Gaussian-filtered original image, in which case it takes the lowest values in the dark areas of the picture, such as the gap between upper and lower jaw or in spaces between teeth. With properly selected initial approximations of these curves and weights α and β it achieves very good segmentation of the image.

The main reason why this approach does not provide the expected results for pantomograms is the same as one of the reasons as to why the previous method was discarded, i.e. severely occluding teeth render it inutile. Dependence on the articulation of gaps between teeth hinders the correctness of the outcome of this method, as a simple Gaussian-filtered image in many cases does not contain distinct enough areas between teeth to ensure a proper segmentation and currently no external force function has been developed to allow for driving the snake in the proper direction in that situation. It should also be noted that snakes work slower than the integral projections, as the initial curve is iteratively improved and both the first- and second-order terms need to be recalculated after every iteration. This issue becomes more pronounced in larger databases, where an additional second added to the calculation time for a single tooth in a small set of 100 images results in almost an additional hour needed for calculation. The last problem stems from the influence of the initial approximation on the final result. No method for selecting the initial curves was presented in [8] and the basis of such selection is not trivial. A preliminary search for the gaps between teeth needs to be conducted before the snakes can be used to separate them and the end result relies on the success of this step.

3 Presented method

3.1 Preliminary steps

To our best knowledge, there are no segmentation methods created explicitly for pantomograms, and the shortcomings of the existing methods encouraged the development of a new approach. It is assumed that before a picture is segmented using the following method, it was enhanced using the method presented in [10]. That method is based on the decomposition of the image into a Laplacian pyramid (that is, a set of images containing downsampled differences between two consecutive layers of a Gaussian pyramid), separating the radiogram into smaller images containing progressively lower frequencies of the signal present in the original image. Then, a range of simple filters is applied to selected layers of the pyramid, including sharpening filter and contrast enhancement methods, before the image is recomposed again. The resulting image has a higher contrast than its original with a slightly increased noise. It is also necessary to locate the gap between the frontal teeth before the segmentation. In this paper a nose position detection is used and then a vertical line on the same position is considered the center, but any other method that provides a valid enough approximation can be used.

3.2 Separating the upper and lower jaw

The first step is the determination of the line separating the upper and lower jaw. The same method as presented in [6] and further described in the previous section is used. In order to automatize the process of segmentation, instead of requiring that the user inputs the initial separating point used by the algorithm, it is selected by choosing the horizontal integral projection with the lowest value, that is also close to the center of the image, usually between 40% and 60% of its height. Because the teeth on the picture create an arc, instead of using the full horizontal line that would pass through teeth further from the incissors, only a small

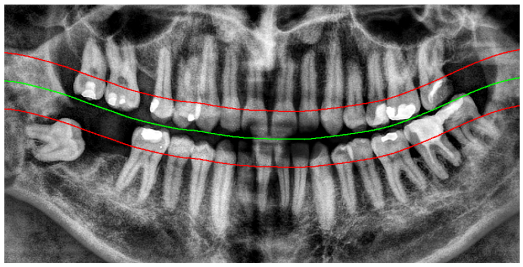


Figure 3: The splines passing through the dental pulp and indicating the height on which the necks of teeth can be found (outer lines) and the spline separating upper and lower jaw (middle line).

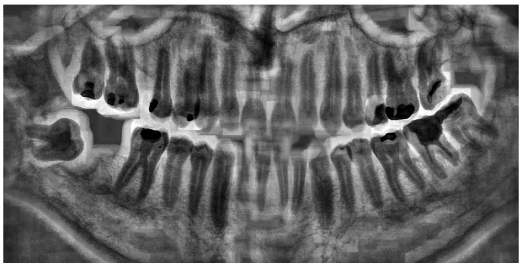


Figure 4: An inverted original image multiplied by range-filtered original image. The size of the neighborhood taken into account during range filtering is 51x51 pixels.

number of pixels (equal to 20% of the width of the image) closest to the previously selected frontal teeth gap is used to calculate the projections. Afterwards the algorithm proceeds as described in [6].

3.3 Localization of the areas between the necks of teeth

The obtained curve is then used to estimate the position of the neck of every tooth, which is the part of the tooth where roots end and the formation of crown and enamel begins. While the crowns of separate teeth tend to occlude with each other and roots are difficult to separate from the underlying bone, the area between the necks of two adjacent teeth is distinguishable enough to be easily found on a pantomogram. Because the necks of teeth are usually on the same height as dental pulp, which is darker than the surrounding teeth, the simplest and fastest method to find a line going through dental necks is to translate the line separating upper and lower jaw vertically, sum the intensities of pixels the line passes through for every translation and select the ones for which there is a distinctive drop of values, indicating the line passes through darker areas of the pulp. This is similar to the integral projections method, but the sums are not calculated along a straight horizontal or vertical line, but instead along a curve that is of identical shape as the spline separating upper and lower jaw and translated vertically. Thus calculated curves for the sample image are shown on fig. 3. In order to make sure neither the original gap between jaws nor a gap between roots of teeth and the edge of the image are selected in lieu of the desired dental pulp line, the vertical translation scope should be chosen to conduct the search for a limited range, automatically discarding translations too close and too far from the line separating jaws. The results of this step are two values, one negative and one positive, that indicate the amount of pixels that the vertical position of every point belonging to the spline separating jaws should be moved for in order to receive the spline that passes through dental pulps of teeth in each jaw.

The next step is the selection of points on each spline representing a gap between the necks of two adjacent teeth. To refine the results of this stage of the algorithm, a new image is created by multiplying the value-inverted original image with local range filtered version of the original image. That image has high values for darker pixels that lie in areas with neighboring points of low and high intensity. The larger the neighborhood used for calculating the range filter, the more distinct those areas become on the obtained image. The resulting image for the sample pantomogram is shown on fig. 4. For both upper and lower jaw, an array of values is saved containing the intensities of points belonging to the splines passing through their respective dental pulps. Sharp spikes on the plot of these values indicate dark spots surrounded by light regions, indicating a gap between necks of teeth. To remove false spikes, the values are Gaussian filtered to smooth the function. Then, starting from the previously selected line separating frontal teeth, small subsets of the values in the array are chosen for comparison. To determine the size of these subsets, average widths of teeth on every position were calculated based on twenty sample pantomograms. Both jaws are fairly symmetrical considering the size of teeth on a given position, thus only eight values need to be calculated for each jaw, one for every tooth from first incissor to last molar.

To ensure that the chosen subsets are wide enough to contain both smaller and larger teeth on a given position, the array values on distances from the previously selected gap ranging from 75% to 175% of the average tooth size are considered as possible positions for the gap. To select the proper spike among these values, for every value in the subset, its closest values are subtracted from it and the results are summed to form a distinction function. This function is then normalized and multiplied by a Gaussian, much like in equation (1). The Gaussian has an expected value based on which direction the algorithm is moving in in a given moment, considering the gap between frontal teeth. The expected value is always chosen to lie in the point indicating the average width of a tooth on the currently searched position, thus if the algorithm is moving

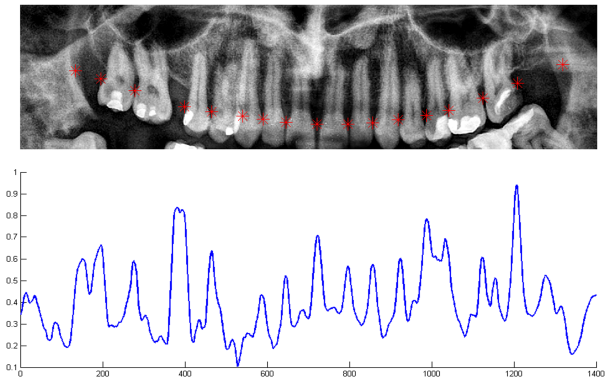


Figure 5: The upper jaw of the exemplary pantomogram with gap locations marked as stars. The plot below shows the intensity values of pixels through which the dental pulp line passes on the image presented on fig. 4, which is searched for spikes indicating gaps between teeth.

left and the chosen subset lies in the distance from -175% to -75% of the average tooth width from the previous gap, then the expected value needs to be chosen in a point that is -100% of the average crown width from the previously selected gap. The function received by multiplying the distinction function and the Gaussian is considered to be the probability function that a given point is the new gap. The argument maximum of that function becomes the new gap position and the beginning point for the next iteration of the algorithm. It works separately for each part of the jaw, i.e. upper left, upper right, lower left and lower right. The algorithm searching for gaps in a given part stops when it has reached the vertical edge of the picture or if it has located eight subsequent gaps, indicating all teeth on that side were found. The results of this step of the algorithm are presented on fig. 5.

Thus calculated gap locations provide a good estimate of the position of areas between teeth. However, in some cases a simple vertical line is not enough to separate two adjacent teeth. Molars and, in some cases, premolars require an additional step of the algorithm to determine the angle of the segmenting line. In order to find a straight line seperating these teeth, an additional point needs to be found between them. A simple greedy algorithm was used, iteratively moving 1 pixel towards the top or bottom of the picture, choosing the pixel in horizontal vicinity with the highest intensity on the inverted and range-filtered image and using it as the basis for the next iteration. After the number of iterations equal to half of the length of an average tooth on a pantomogram, the position of the last result becomes the second separating point and the line passing through gap between necks of teeth and the second separating point becomes the segmentation line. The drawback of this approach of finding the second separating point is that the roots of the tooth need to be distinctive enough from the background and neighboring teeth, so using that algorithm with incissors and canines usually produces incorrect results. The separating lines acquired using this algorithm on a sample pantomogram are presented on fig. 6.

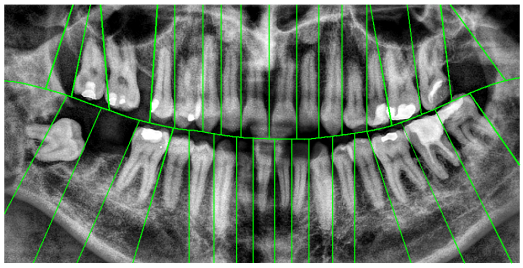


Figure 6: Lines separating individual teeth on the pantomogram. Simple vertical lines are used for incissors and premolars, while rotated lines achieved by determining two points between the teeth are used for molars.

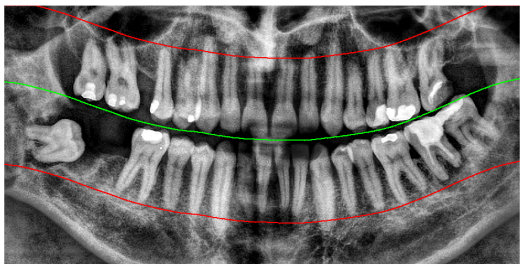
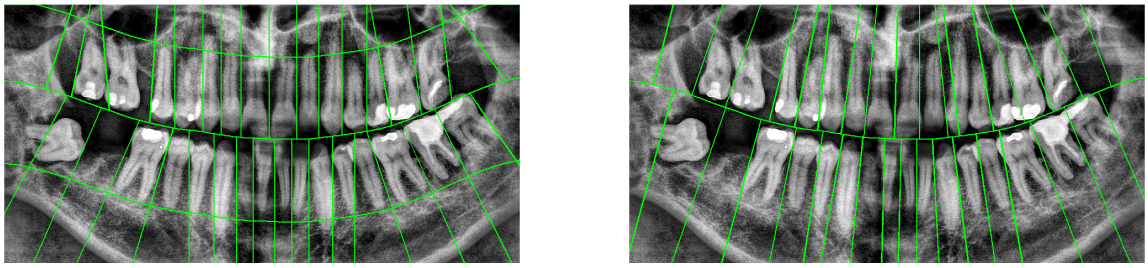


Figure 7: The splines that determine the end of roots (outer lines) and the spline separating upper and lower jaws (middle line) on which they are based.

3.4 Removing the areas below the roots of teeth

The last step is to remove the areas below the roots of teeth. The method used to detect where teeth end is similar to detection of dental pulp line, i.e. the curve separating both jaws is translated vertically in search of an alignment where the sum of pixels it passes through is lower than the surrounding results, indicating the area between the teeth line and the cheekbone line was achieved. The only difference between this part of the algorithm and the search for the line of necks of teeth is the range of translations considered in the search. The results achieved using this algorithm to determine the curve separating teeth from areas below roots are presented on fig. 7. After finding these separating lines, every segment of the picture lying between four curves (jaws gap line, two consecutive lines separating adjacent teeth and the line below the dental roots) is considered an area possibly containing a tooth and can be later used in the process of feature extraction.



(a) (b)

Figure 8: A comparison of the result of the proposed method (8(a)) and an another popular method used in the segmentation of dental radiograms - integral projections (8(b)).

4 Results and discussion

The above described method was tested on a database of 218 pantomograms, each sized 1500x800 pixels, and some exemplary results are presented in this section. Firstly, fig. 8 shows a comparison of results achieved for the proposed method and the integral projections method. The proposed method offers better results with more compact segments and less segmentation lines passing through the areas of teeth. It should also be noted that while the calculation of the plot shown on fig. 5 takes less than 1 second, the calculation of the integral projections in every point of the jaws separation curve takes 212 seconds on the same computer.

Some additional test results can be seen on fig. 9. If the teeth can be separated using a single line, the algorithm is usually able to find the optimal line of separation. The described method is also able, in some cases, to separate unerupted or not fully erupted teeth, as can be seen on fig. 9(c). The obvious bad results can be found in cases of occlusions so severe that it is impossible to find a straight line to separate both teeth, such as in the case of the incissor and the first premolar (respectively third and fourth tooth from the center of the image) of figures 9(a) and 9(e). Malaligned teeth can also be separated to a degree, but in the case of severe problems in alignment the separating line can not capture the whole tooth within a segment, for example the furthermost bottom left molar in figure 8(a). Two neighboring dental fillings can also be problematic, as they are the brightest areas of the image and blend easily, making it impossible to decide where one tooth ends and the second begins (like the molars in the lower right region of figure 8(a)). Detection of the ends of dental roots helps in removing bright areas of the underlying bone that would be otherwise attributed to the tooth, but in some cases a fragment of the tooth is removed too, like on figure 9(e). The last problem stems from the fact that dental pulp can easily be mistaken for the edge of the tooth, resulting in a mis-segmentation. This happens in the case of the lower left incissor (third tooth) on figure 9(b), where the separating line passes through the middle of the tooth, but because average teeth widths are used, the next two teeth segments are wider and the final eighth tooth is separated correctly.

5 Conclusions and future work

In this paper a novel method for segmenting dental panoramic radiograms into regions containing single teeth was described. It uses a different approach than the existing algorithms developed for intraoral images, focusing on detecting gaps between necks of teeth and roots of teeth, that are both easy to find on a pantomogram and allow to separate teeth even in the situation of occlusions. The presented method works fully automatically and with speed comparable to methods developed for intraoral images.

A considerable improvement of its results could be achieved if a more sophisticated method was used instead of the greedy algorithm to determine the second point through which the separating line between two teeth is traced. The method could also be used as an initial step for further segmentation with methods such as active contours to provide improvement of the results. Finally, the process of removing the areas below the roots of teeth could be conducted separately for every tooth instead of for all teeth in a jaw simultaneously, thus reducing the amount of root fragments incorrectly segmented outside of the tooth area.

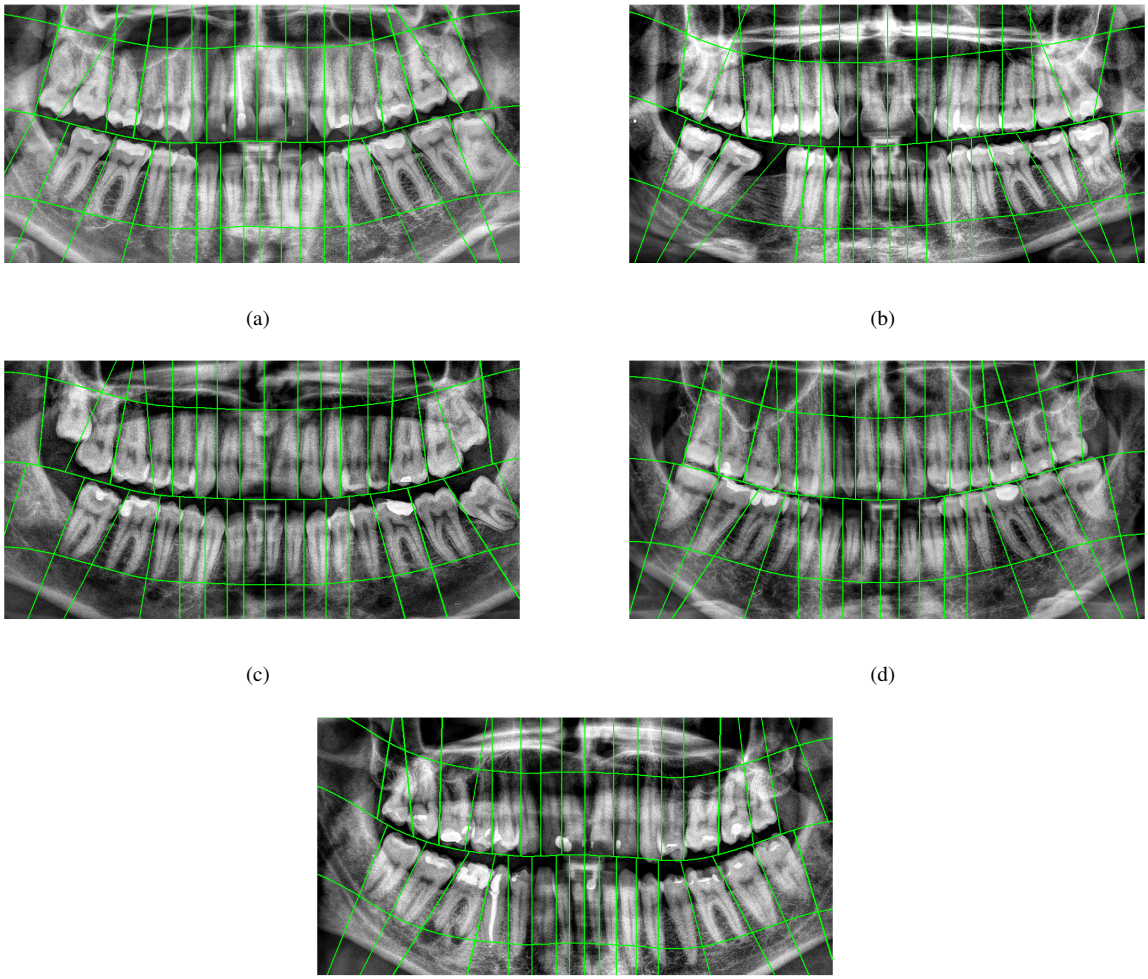
Acknowledgements

I would like to thank my parents for their continuous and discrete support during the writing of this article.

References

[1] S. Lee et al. The diversity of dental patterns in orthopantomography and its significance in human identification. *J Forensic Science*, 49(4), 2004.

[2] D.E.M. Nassar, H.H. Ammar. A prototype automated dental identification system (ADIS). In *Proceedings*



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

Figure 9: More exemplary results of the proposed method.

of the 2003 annual national conference on Digital government research, 2003.

[3] M. Abdel-Mottaleb et al. Challenges of developing an automated dental identication system. In *IEEE Mid-west Symposium for Circuits and Systems, Cairo, Egypt*, 2003.

[4] C. Michael Bowers. *Forensic Dental Evidence*. Elsevier, 2004.

[5] A.K. Jain et al. Toward an automated dental identification system(ADIS). In *Proc. Int. Conf. Biometric Authentication, Hong Kong, China*, 2004.

[6] A.K. Jain, H. Chen. Matching of dental x-ray images for human identification. *Pattern Recognition*, 37(7), 2004.

[7] A. Said et al. Dental x-ray image segmentation. In *SPIE Technologies for Homeland Security and Law Enforcement conference*, 2001.

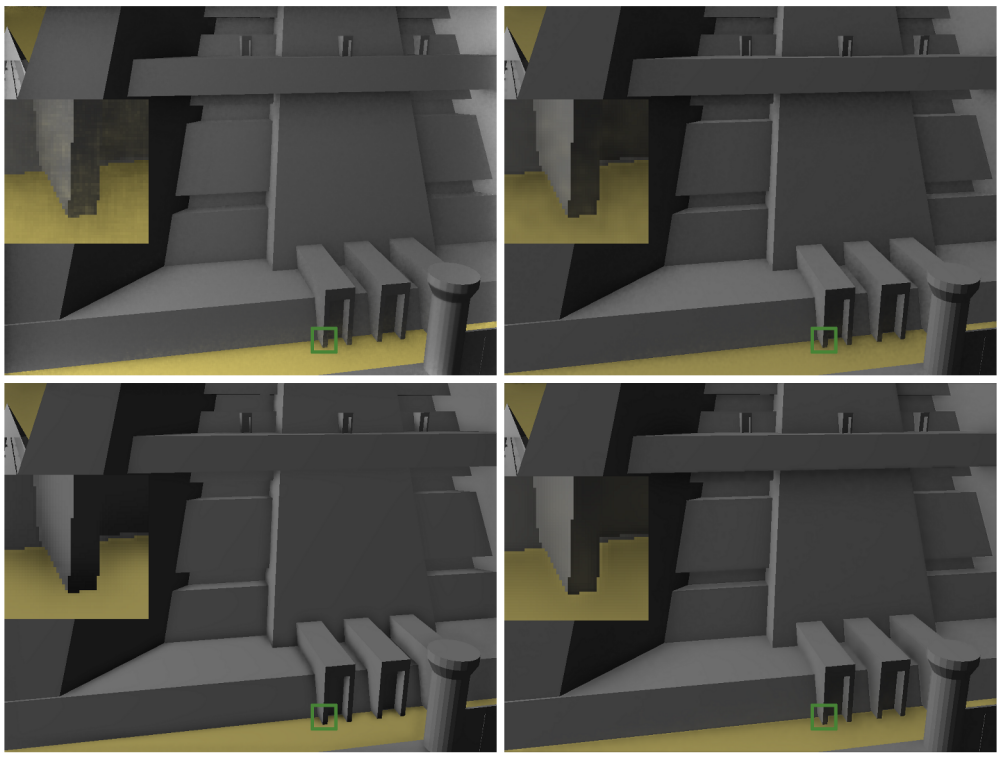
[8] J. Zhou, M. Abdel-Mottaleb. A content-based system for human identification based on bitewing dental x-ray images. *Pattern Recognition*, 38(11), 2005.

[9] M. Kass, A. Witkin, D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4), 1988.

[10] D. Frejlichowski, R. Wanat. Application of the Laplacian pyramid decomposition to the enhancement of digital dental radiographic images for the automatic person identification. In *Image Analysis and Recognition 7th International Conference*, 2010.

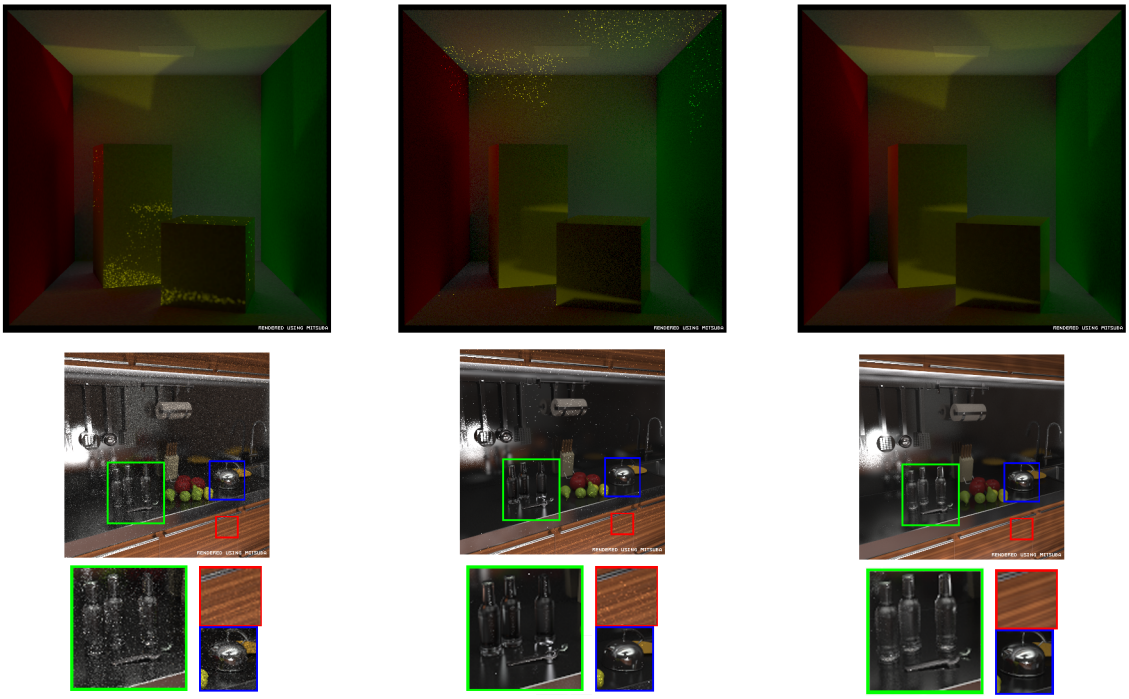
Color Plates

István Szentandrási
Modern Methods of Realistic Lighting in Real Time
pp. 17–24



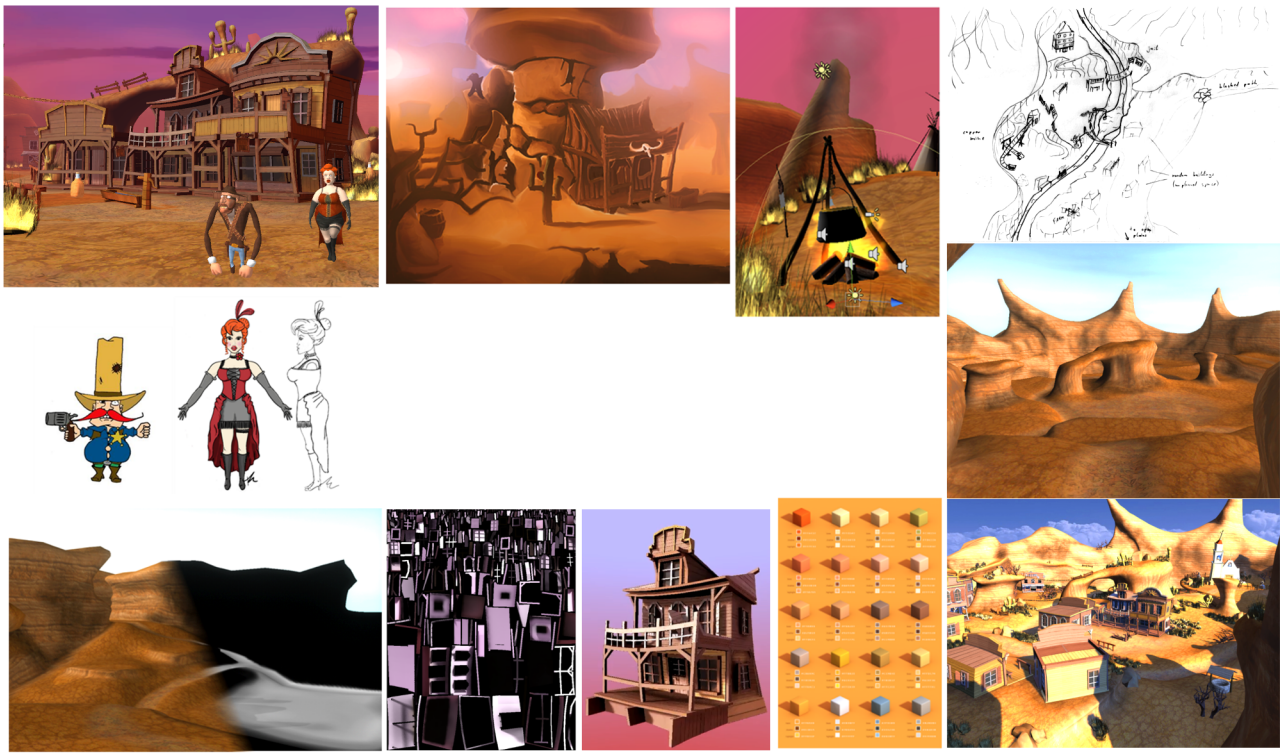
Top left - SSDO computed using two steps. Top right - SSDO computed using modulation factor in one step. Bottom left - SSAO. Bottom right - SSDO computed using modulation factor, subsampling and preprocessing. Regions with the green border are shown in more detail on the left side of the images.

Jiří Vorba
Bidirectional Photon Mapping
pp. 25–32



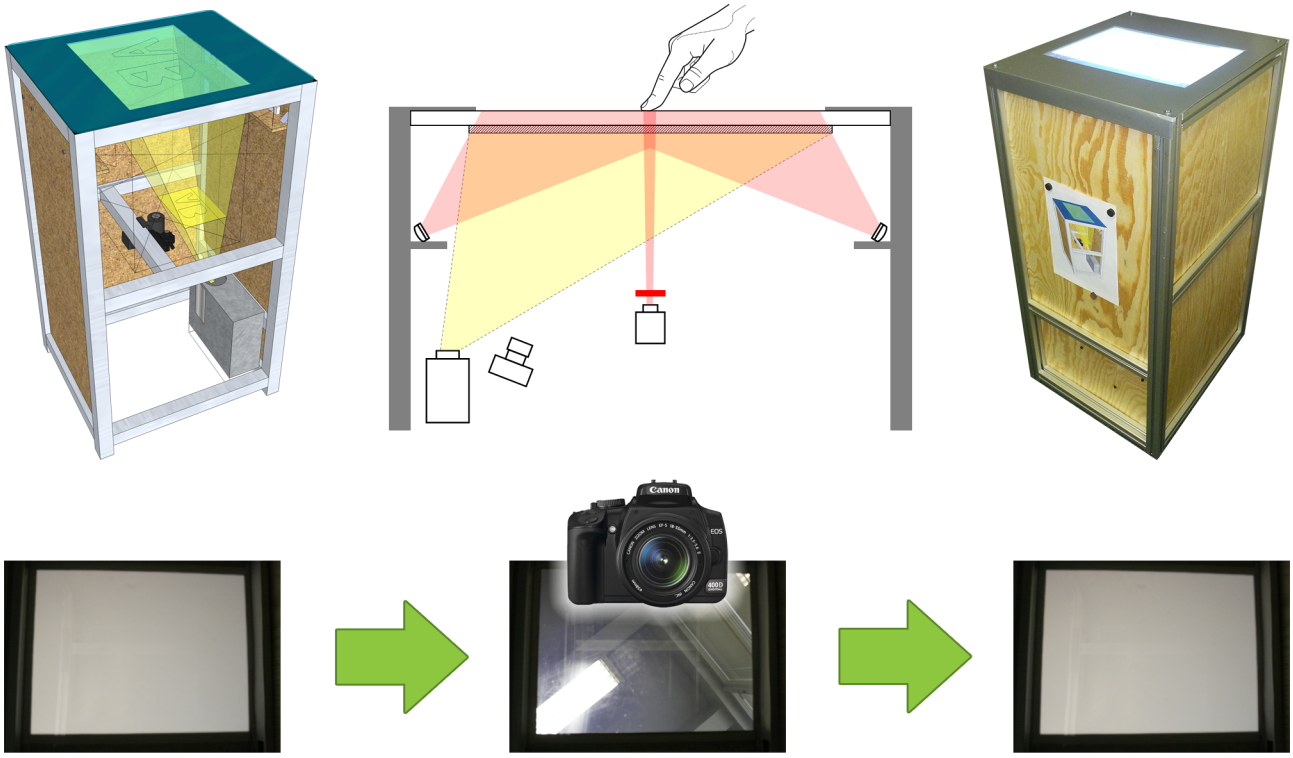
Top: (from left to right) Cornell box like scene rendered by PM without final gather, with final gather, MIS combination of both strategies by our BDPM method.
Bottom: (from left to right) Images rendered by PT (30.4h), PM with final gather (4h), our BDPM method (4.1h).

Labschütz Matthias et al
Content Creation for a 3D Game with Maya and Unity 3D
pp. 75–82



In clockwise order: 1: in-game screenshot, 2: concept, 3: effects, 4-6: level stages, 7: color table, 8: lighting, 9: multitexturing, 10: character concept.

Jakub Hušek
Multi-touch Table with Image Capturing
pp. 91–98

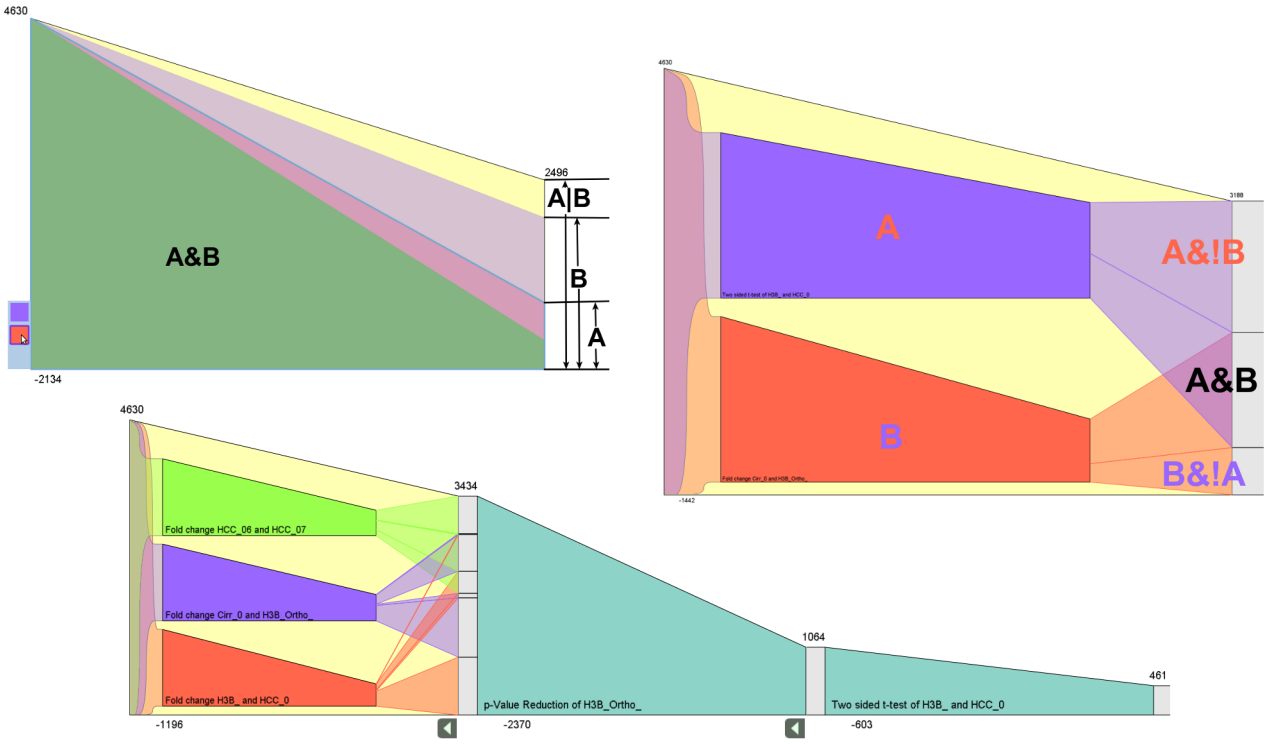


Left: a sketch, middle: a scheme, right: a photo of the final prototype. Down: a switchable diffuser in a diffusing and clear state (a photo can be taken).

Peter Mindek
Maximum Intensity Projection Weighted by Statistical Cues
pp. 149–155

Visualizations of an abdominal CT scan using MIP weighted by statistical cues with different settings.

Thomas Geymayer
Visualizing the Effects of Logically Combined Filters
pp. 157–163



Top: Two filters combined with logical OR visualized once with stacked sub-filters (left) and once with parallel sub-filters (right). Bottom: Advanced filterpipeline starting with three filters combined with a logical OR followed by two more filters.

Sponsors of CESCG 2011

DCGI

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

Computer Graphics and Human Computer Interaction Courses 2010/2011

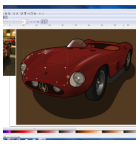
Bachelor´s degree

Computer Graphics

Introduction to graphics APIs, programming simple interactive graphics applications.

Multimedia 1

Methods and technologies for multimedia content production and processing.



Multimedia a Graphics Applications

Overview of selected multimedia and graphics applications and their usage in practice.

3D Modeling and Virtual Reality

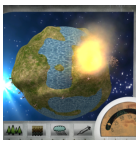
Overview of Virtual Reality technology and in depth study of Virtual Reality Modeling Language with a focus on efficient design of interactive environments.

Mobile Applications Development

Properties, advantages, and limitations of mobile technologies. Embedding mobile devices into complete information systems.

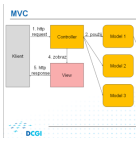
Testing of User Interfaces

Principles of design and testing of user interfaces. Specialized user interfaces (handicapped users, user interfaces for mobile devices).



Computer Games and Animations

Computer games design and programming. Representation, animation and rendering of geometrical models, collision detection, shader programming.



Web Applications Development

Design of web presentations, scripting on the client side, creation of a dynamic web applications on the server side (XHTML, CSS, JavaScript, PHP).

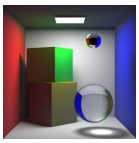
Course of Multimedia Applications

Practical course of Maya modeling toolkit.

Graphics Design

Principals of graphics design and typography. Individual design of electronic document. An important part of the course is painting.

Master´s degree

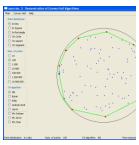


Algorithms of Computer Graphics

Modeling and rendering of graphics primitives in 2D and 3D, visibility determination, color models, image representations, basic photo-realistic rendering algorithms.

User Interface Design

Formal description of user interfaces, user models, fundamentals of perception, cognition.



Computational Geometry

Efficient algorithms for determining properties and relations of geometric entities (geometric search, point location, convex hull construction, nearest neighbors).

Realistic Image Synthesis

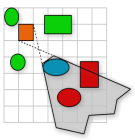
Global illumination methods used for photo-realistic rendering such as raytracing, Monte Carlo path tracing, or photon maps.

Multimedia and Computer Animation

Overview of contemporary problems in computer animation. Focus on raster animation (video processing) and vector animation (motion modeling and motion control of rigid body models).

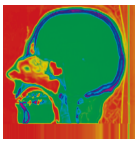
Intermedia Design and Technologies

Technological and artistic concepts in multimedia put together. The course is organized together with art universities.



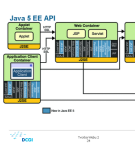
Data Structures for Computer Graphics

Fundamentals of data structures commonly used in computer graphics. The course focuses on multidimensional data used to represent 3D scenes.



Visualization

Theoretical background of visualization and the application of visualization in real-world examples. Revealing hidden dependencies in the data, human perception.



Development of Web Applications 2

Description and practical usage of servers and frameworks for development of web applications. Integration of servers, databases, and frameworks.

Computer Graphics 2

Programming of advanced graphics techniques using OpenGL and its extensions. Using GLSL shading language for GPU rendering.

General Purpose GPU Programming

Basic principles of GPGPU programming, parallel programming concepts, CUDA, code optimizations.

Psychology in HCI

Overview of psychological findings applicable in HCI. Desing of interactive software components taking into account psychological properties of the user.

http://dcgi.felk.cvut.cz

For further information about studying at our faculty please contact us by email: office@dcgi.felk.cvut.cz
Department of Computer Graphics and Interaction, Karlovo náměstí 13, 121 35 Praha 2, Czech Republic

Univ.Prof. Dr. BERT JÜTTLER
Institut für Angewandte Geometrie

Tel.: +43 732 2468-9178
Fax: +43 732 2468-29162
bert.juettler@jku.at

Sekretärin:
MONIKA BAYER
DW 9162
monika.bayer@jku.at

Linz, 4. April 2011

Open positions for PhD students

Several positions for PhD students are currently available at the Institute of Applied Geometry (www.ag.jku.at) at Johannes Kepler University of Linz, Austria (www.jku.at).

Starting date: September 2011 or later.

Required scientific background: Applied Geometry, in particular Computer Aided Geometric Design.

Remuneration: The salary follows the guidelines of the Austrian Science Fund (currently 1.877,40 Euro per month, paid 14 times per year).

Contact and further information: Send an email to bert.juettler@jku.at.

In order to apply, please send a short Curriculum Vitae (1-2 pages) and an application letter by email to Bert Juettler.

Be the big thinker behind tomorrow's technology

Qualcomm is taking its traditional strengths in digital wireless technologies in exciting new fields such as Augmented Reality and Computer Vision.

Join Qualcomm's new Corporate R&D team in Vienna, Austria, to contribute to the development of a new generation of wireless devices using Augmented Reality.

- Be part of a team developing mobile Augmented Reality enabling technology.
- Design, implement, and verify algorithms for Augmented Reality
- Develop and implement new computer vision approaches for mobile devices

Qualcomm Austria Research Center

Augmented Reality Software Engineer

Skills/Experience:

The ideal candidate has outstanding C++ and object-oriented programming skills, as well as five or more years of experience in at least two of the following sub-topics:

Augmented Reality

- Experience with hands-on laboratory implementation, verification and optimization of augmented reality systems
- In depth understanding of natural feature based AR systems
- Familiarity with coherent rendering techniques for AR
- Familiarity with AR user interfaces
- Experience with outdoor AR systems is a plus

Computer Vision

- Experience with hands-on laboratory implementation, verification and optimization of real-time monocular computer vision systems.
- In depth understanding of state of the art natural feature detection and description algorithms like SIFT, SURF, and FERNS
- In depth understanding of 2D and 3D tracking algorithms
- Experience with SLAM algorithms
- Experience with face detection or OCR is a plus

Education:

Master's or PhD (Computer Science)

Ready for a challenge?

Please apply directly on our website at www.qualcomm.com/careers/ searching by job title or requisition number **G1882475**.

Be the big thinker behind tomorrow's technology

Qualcomm is taking its traditional strengths in digital wireless technologies in exciting new fields such as Augmented Reality and Computer Vision. Join our new Corporate R&D team in Vienna, Austria and interact closely with system engineers and software developers in a world-class AR team.

- Be part of a team developing mobile Augmented Reality enabling technology
- Design, implement, and verify algorithms for Augmented Reality
- Support development of new computer vision approaches for mobile devices

Qualcomm Austria Research Center

Augmented Reality Engineer Intern

Qualcomm offers flexible start dates during 2011 for Intern Engineers, a very competitive salary, holiday and sick pay, and where applicable assistance with accommodation. Assignment duration ~4 months. Students should be mid-study looking for work experience or a mandatory internship to contribute towards completing their degree.

Skills/Experience:

The ideal candidate has outstanding C++ and/or Java programming skills, expertise in object-oriented design, and a good foundation in one of the following ***areas***

Augmented Reality & Computer Vision

- Familiarity with camera capturing, image processing, and rendering
- Basic understanding of natural feature detection and description algorithms
- Familiarity with 3D Interaction Techniques preferably related to AR

Integration & Verification

- Exposure to laboratory implementation, verification and optimization of computer vision systems or augmented reality systems
- Ability to create test environments using software and hardware tools

Software Tools

- Familiarity with software development environments
- Interest in developing various stages of the tool pipeline

Education:

BS or MS student in Electrical or Computer Engineering (preferred) or Computer Science. Please include graduation date and grades in your resume.

Working Location:

Vienna, Austria

Please apply directly on our website at www.qualcomm.com/careers/ under requisition number **E1879350**.

Be the big thinker behind tomorrow's technology

Qualcomm is taking its traditional strengths in digital wireless technologies in exciting new fields such as Augmented Reality and Computer Vision. Join Qualcomm's new Corporate R&D team in Vienna, Austria, to contribute to the development of a new generation of wireless devices using Augmented Reality.

- Be part of a team developing mobile Augmented Reality enabling technology
- Drive software integration and tools to general high quality feature-complete software for mobile platforms
- Verify functionality and interfaces of SW modules
- Develop and implement tools required for verification and automated testing

Qualcomm Austria Research Center

System Integration Engineer

Skills/Experience:

The ideal candidate has outstanding skills in the following areas:

- Analysis and troubleshooting of technical problems
- Integration and verification of real-time SW modules (based on at least 3 years of relevant work experience)
- C/C++ programming skills
- Scripting skills (Python, Perl)
- Candidates should be flexible in their work assignments as priorities can change quickly in this fast paced environment.
- Candidates must be able to evaluate technologies, systems, and devices through testing, logging and analysis
- Excellent written and verbal communication skills
- Must be familiar working with and testing applications on AndroidOS, Linux, Windows XP, and Windows 7

Responsibilities

- Drive SW integration of Augmented Reality SDKs
- Troubleshoot and debug using Windows, Linux and Android tools
- Reproduce and document issues
- Create testing environments using software and hardware tools
- Develop test procedures, execute tests, and isolate problems
- Identify areas for integration tools and specify requirements

Education:

Master's Computer Science

Please apply directly on our website at www.qualcomm.com/careers/ searching by job title or requisition number **E1883539**.

GMSV
Geometric Modeling and Scientific Visualization Center

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Geometric Modeling and Scientific Visualization Center

King Abdullah University of Science and Technology

The Geometric Modeling and Scientific Visualization (GMSV) Center serves as a focal point for interdisciplinary research, encompassing modeling, analysis, algorithm development and simulation for problems arising throughout various fields including energy, environment, biosciences, earth sciences, materials science, fluid mechanics and marine science. The center also functions as a crossroads for industry researchers working on similar multiscale modeling and simulation problems.

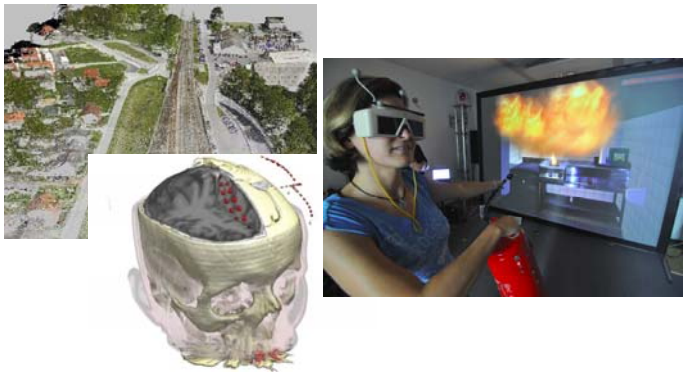
Vision and Focus

The GMSV Center faces the challenge of becoming one of the world's leading research centers in computer graphics, geometric modeling and visualization. It seeks to contribute in a seminal way to both the modeling and visualization community as well as to the body of science as a whole.

We continue to develop collaborations with world class universities and research institutes, including KAUST's own cadre of top scientists. We have an excellent founding faculty, associates and students, and our visualization facilities are arguably the best in the world.
The GMSV Center ensures that KAUST is at the forefront of computer graphics, modeling, visualization and immersive environments and petascale computing. The center possesses top expertise in geometry processing and modeling, computational geometry, high performance visualization, rendering, and imaging science. The GMSV Center has a strong focus on interacting with other engineering and scientific efforts on campus and with our research partners, as well as on novel applications, which have a high potential for the advancement of science and technology.

The GMSV Center is currently seeking highly motivated and creative researchers at the M.S., Ph.D., post doc and research associate levels. It is open to new ideas and innovative thinkers. If you are one of these people please contact: gail.redman@kaust.edu.sa

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

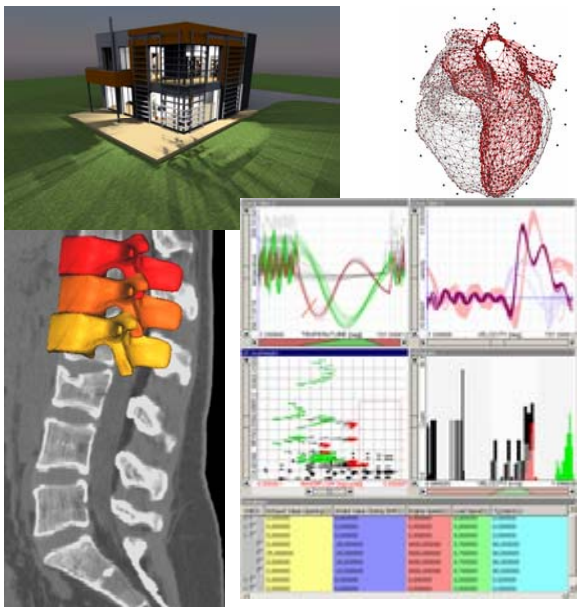
Scientific Partners of VRVis:

- 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.